Frontmatter

Editors

SIGAPP FY'18 Quarterly Report

J. Hong

Selected Research Articles

Usability Evaluation Framework for Domain-Specific Language: A Focus Group Study

I. Poltronieri, A. Zorzo, M. Bernardino, and M. Campos

Adaptive Information Distribution for Dynamic Sets using Multicast Push and Pull

M. Prellwitz, H. Parzyjegla, and G. Mühl

Functional Feasibility Analysis of Variability-Intensive Data Flow-Oriented Applications over Highly-Configurable Platforms

S. Lazreg, P. Collet, and S. Mosser

An Effective Sequence Structure Representation for Long Non-Coding RNA Identification and Cancer Association using Machine Learning Methods

M. Madhavan and G. Gopalakrishnan
SIGAPP FY’18 Quarterly Report

July 2018 – September 2018
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 and Location</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 Lencastré</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 December 2018.
Usability Evaluation Framework for Domain-Specific Language: A Focus Group Study

Ildevana Poltronieri  
Pontifical Catholic University of Rio Grande do Sul - PUCRS  
Av. Ipiranga, 6681  
Porto Alegre, Brazil  
ildevana@gmail.com

Avelino Francisco Zorzo  
Pontifical Catholic University of Rio Grande do Sul - PUCRS  
Av. Ipiranga, 6681  
Porto Alegre, Brazil  
avelino.zorzo@pucrs.br

Maicon Bernardino  
Federal University of Pampa - UNIPAMPA  
Av. Tiaraju, 810  
Alegrete, Brazil  
bernardino@acm.org

Marcia de Borba Campos  
Pontifical Catholic University of Rio Grande do Sul - PUCRS  
Av. Ipiranga, 6681  
Porto Alegre, Brazil  
marciabcampos@hotmail.com

ABSTRACT
Software engineers are increasingly taking advantage of new methods to improve software quality. The use of languages developed for specific domains, which in the literature are known as Domain-Specific Languages (DSLs), has grown in the past years. Although several experimental studies that subjectively evaluate usability of these languages can be found in the literature, few of them have taken advantage of applying Human-Computer Interaction (HCI) techniques in those evaluations. Therefore, the main goals of this paper are to present a usability evaluation framework for DSLs, called Usa-DSL, and to show the evaluation of the framework through a Focus Group method. The evaluation was performed by seven specialists that discussed the framework usability and suggested some modifications of our initial proposal. The specialists recommendations were incorporated in the final framework presented in this paper.

CCS Concepts
•Software and its engineering → Domain-specific languages;  
•Human-centered computing → Usability testing;  
•General and reference → Focus group; Empirical studies;

Keywords
Usability evaluation, Usability testing, Domain-specific languages, Focus Group

1. INTRODUCTION
Usually, General-Purpose Languages (GPL), such as Java, C#, Ruby, Python, among others, are used for software development. On one hand, this variety of programming languages, allied with the complexity of several applications, may present several difficulties regarding system modeling, implementation, evaluation and maintenance. This may cause different problems and also compromise the quality of the developed systems. On the other hand, there are domain-specific applications that may benefit from languages with specific characteristics, which contribute to the increment of performance, representation, business domain abstraction, better communication between developers and business analysts, among others aspects. Hence, through the development of different languages, software engineers try to facilitate the knowledge sharing of certain domains.

Languages used to describe characteristics of certain domains are called Domain-Specific Languages (DSLs) [14]. Currently, DSLs have been applied to several different domains. For example, there are DSLs applied to software architectures anomalies [1] or performance testing [7] [8]. The difference among these DSLs are defined by theirs syntax and semantic, which are determined by the problem domain. It is important to mention that several different DSLs can be used to represent a domain in order to model its characteristics, without necessarily overlapping them.

Despite all the benefits of DSLs, there is still some effort needed to develop DSLs. Therefore, it is important that these languages meet several usability and satisfaction criteria related to the user experience [32]. Meeting these criteria will enable users to use these languages in a more independent and easier way. This is even more important if someone considers the existing diversity of domains and contexts in which DSLs can be applied to. Furthermore, users (i.e. software engineers) satisfaction is an important criteria that has to be taken into account when developing a DSL [24].

Therefore, considering different DSL concepts, or even different domains in which DSLs are applied to, this paper presents a framework to evaluate DSLs usability: the Usability Evaluation of Domain-Specific Languages (Usa-DSL) framework. This framework takes into consideration Human-Computer Interaction (HCI) aspects and apply them to the
evaluation of DSLs usability. This paper also presents the framework evaluation based on a Focus Group [18, 20].

This paper is organized as follows. Section 2 presents some background on the subjects related to this work. Section 3 discusses the related work. Section 4 presents the Use-DSL framework, as well as motivations for building this framework. Section 5 introduces the Focus Group, describing the pilot instrument as well as the planning, preparation, moderation and data analysis phases of the empirical evaluation. Finally, Section 6 presents the conclusion and future work.

2. BACKGROUND

Software Engineering (SE) [33] is the area that comprises all stages of software development, adopting systematic and organized approaches to produce high quality software efficiently. The adopted approaches involve the analysis of practical issues such as cost, time, reliability and the customers needs. In SE there are different methods and techniques that can be applied depending on the type of application being developed.

One of the processes available in the literature is the Domain Engineering used in the development of reusable applications, which is intended for modeling and identification of the characteristics of an application domain [23] [36]. Its main goal is to allow same domain systems to be built from common processes and artifacts, reusing concepts and resources [26]. The Domain Engineering process allows, from the execution of its activities, the results to be obtained as design patterns, application generators, reusable components, reference architectures or Domain-Specific Languages (DSL), a.k.a. Domain Specific Modeling (DSM) [17].

The DSM is an SE methodology, which has a domain analysis phase that defines the rules, resources, concepts and properties of the domain that must be identified. Therefore, DSM as well as transformation engines and generators are technologies that combined provide support to develop Model-Driven Engineering (MDE) [29]. MDE enables systems to be developed and tested based on domain modeling. It allows to generate code or to analyze its model automatically [10].

In order to minimize the difficulties in applications development, one of the possibilities is the use of Model-Driven Development (MDD) [31]. MDD allows the automatic code generation based on systems models and is based on DSLs. A DSL is a type of programming language or specification used in software development and Domain Engineering [14]. This type of language has as purpose the construction and notation adapted to a given domain [19], i.e. the fundamental difference between a DSL and a General Purpose Language (GPL) is that DSL is constructed from the domain of the problem and not the domain of the solution.

DSL as other programming languages has syntax, which defines its structure, and semantics, which defines its meaning. One of the important characteristics in a DSL is the form of the adopted representation, since it must be in accordance with the concepts of the domain that is being modeled. Thus, the principle of representational fidelity [17] [35], which states that only one form represents each concept of the domain, not only simplifies the definition of the notation, but also ensures that all concepts will be represented in this language. The semantics of a DSL defines the meaning of each language construct, and each element has a meaning determined by that domain.

There has been a lot of effort to create and to use DSLs as a resource to facilitate system construction, to increase productivity and to ease its maintenance. However, since a DSL deals with the problem domain, and not with the solution domain, users (software engineers, programmers, ...) not always accept them [15]. This might be mitigated if usability criteria were used during the development of a DSL.

Usability intends to ensure that interactive systems are easy to use, easy to learn, efficacious and pleasant to use, from the users perspective [25]. Usability is not only related to user interface appearance, but refers, mainly, to how the systems interact with users. Therefore, to adopt usability criteria brings several advantages to a user in terms of productivity, better job quality, and satisfaction. To a system producer it brings reduction in terms of maintenance, meets the user expectation, reduces final training costs, and increases competitiveness. Hence, usability should be considered in the whole development cycle of a DSL, improving during DSL lifetime, in order to meet users expectations.

Usability studies started at the beginning of 1980's and is not yet systematized by the developers of interactive systems, including developers of DSLs. There is a lack of usability evaluation, or lack of reporting, during the development of DSLs [27]. Some authors mention that industry does no invest in DSL evaluation since there is no experimental evidences that show qualitative improvement in the development of DSL when a systematic DSL usability evaluation is produced [15]. Furthermore, they also consider that, since a DSL is produced interacting with domain specialists, it usually meets the usability criteria. However, the domain specialists might not be the end users, and therefore, they might be biased by their own experiences, compromising the final DSL usability.

To integrate methods, activities and artifacts from Usability Engineering in organizations that already use a process to develop systems is not simple. Nebe et al. [21] discuss the similarities and differences from Software Engineering and Usability Engineering, identifying patterns and processes from these two areas, and point out some fragility from Software Engineering modeling processes in relation to Usability Engineering. Nonetheless, there has been an increase in the application of usability techniques in software development, even though this is not an easy task [13]. One of the reasons might be that software engineers and usability engineers have different views on how to develop a software system, and, therefore, some conflicts might arise due to procedures or terminology differences.

Recent studies on usability evaluation show that usability has been considered during the development of DSLs [27]. Section 3 shows some of these studies. The lack of integration on Software Engineering and Usability Engineering have also been reported by those studies.
3. RELATED WORK
In a previously developed study [27], a Systematic Review was performed and presents a taxonomy for DSL usability evaluation (see Figure 1). The main research described in that paper, and that are related to this work, are presented in Table 1. The next paragraphs will summarize the contributions of the papers presented in Table 1.

Albuquerque et al. [1] presented an evaluation method called Cognitive Dimensions Notation (CDN) that contains 14 dimensions. Such dimensions served to support the development of the characteristics of their work, i.e., DSL expressiveness, which refers to what extent the DSL represents the domain, and DSL conciseness, which refers to what terms can be deleted without compromising the domain artifact representativeness. These characteristics were also divided into metrics such as: expressiveness, which is composed of hidden dependencies, abstractions, mapping proximity; and, conciseness, which is composed of viscosity, visibility, diffusion and hard mental operations.

Barisic et al. [4] suggested that for usability evaluation it is important to first define the usability requirements. Each requirement is assessed by a set of quantitative metrics using Goal Question Metric paradigm (GQM). Regarding cognitive aspects, Barisic et al. [6] performed a controlled experiment with six participants to evaluate a cognitive model to languages based on user scenarios. The cognitive activities, in which in the language are: syntax and semantic learning, syntax composition needed to fulfill a role, syntax understanding, syntax debugging, and changing a function that was written by any developer.

Although Ewais and Troyer [12] did not explicitly describe an evaluation method, they used a strategy to evaluate the usability of a language before it would be implemented. To perform this evaluation, fourteen subjects participated in an experiment to evaluate the use of visual domain specific modelling languages for designing.
Different from other studies, Barisic et al. [5] presented the analysis of four controlled experiments. The authors mentioned that the usability evaluation performed in each experiment was based on users interviews, open questionnaires, testing using tools support and multiple-choice questionnaires. Barisic et al. [3] used a recommendation-based methodology that considers user-centered techniques. The main activities that their methodologies describe are: domain analysis, language design, controlled experiment as design, development process, and deployment.

Sinha et al. [32] based their evaluation on four heuristics proposed by Nielsen, and for each heuristic there was a set of metrics. On one hand, learnability was measured through the number of errors a subject made, divided by effort; while efficiency was measured by the size of the test set divided by effort. On the other hand, satisfaction was measured in four levels: frustrating, unpleasant, pleasant, and pleasurable. Therefore, it was possible to have a quantitative evaluation of a DSL when analyzing its usability.

From other point of view, Seffah et al. [30] mentions the obstacles that occur to the stakeholders roles on the development process, arguing that the terms “friendly user interface” and “user interface” are obstacles to interactive and usable systems. The author points out that the behaviour of both communities illustrates the separation, isolating the user interface from the rest of the system.

Another important studies is presented by Alonso-Rios et al. [2] that describes a usability taxonomy. This proposed taxonomy helped to support the development of the Usa-DSL framework, once many attributes shown in the authors’ taxonomy, from the perspective of system usability, were dealt with in our proposal.

Although several researchers have presented some ideas on how to evaluate DSLs, all of them evaluate DSLs in an ad hoc manner. The first ideas related to developing a new framework to evaluate DSLs usability were presented in a previous work [27]. Regarding the techniques and methods, some studies present the adaptation or use of a set of usability metrics. Despite the efforts in previous research, there is still a lot of work to transform the conception of DSLs into an easier and more comprehensible and expressive task in relation to the domain that they intend to represent. In addition, it is also necessary to develop processes, methods and techniques that assist in the usability assessment of DSLs.

Section 4 presents the description of the Usa-DSL framework. The current framework includes suggestions from a Focus Group that was applied to evaluate the framework (see Section 5).

### 4. USA-DSL FRAMEWORK

In order to understand how, usually, DSL designers evaluate DSL usability, a Systematic Review [27] was performed to analyze studies that apply HCI concepts [22, 25, 34] in their evaluation (see Section 3). As mentioned before, different studies presented some discussion on how to use usability concepts to evaluate a DSL, however, to the best of our knowledge, no framework or method to perform usability evaluation of DSLs had yet been proposed. Therefore, this section presents a framework to evaluate DSL usability, called Usability Evaluation for Domain Specific Languages framework (Usa-DSL).

The next subsections present the Usa-DSL structure and the details about its phases, steps and activities.

#### 4.1 Usa-DSL Structure

The Usa-DSL framework structure is based on the project life cycle process [34], which is composed of phases, steps and activities (see Figures 2 and 3). Basically, Usa-DSL is organized in phases, in which a set of steps has to be taken. For each step in a phase, there is one or none activity that has to be executed. Notice that some steps, in certain phases, have no activities, e.g. step “2 - Ethical and Legal Responsibilities” in phase Analysis has no activity, while this same step in phase Execution has activity “E2 - Introduce the Form and Collect Signatures of Subjects”.

There are four phases in the Usa-DSL framework: Planning, Execution, Analysis and Reporting (PEAR phases).

Each phase can be split into a set of the following steps: 1 - Evaluators Profiles; 2 - Ethical and Legal Responsibilities; 3 - Data Type; 4 - Empirical Study Method (SE); 5 - Evaluation Method (HCI); 6 - Metrics; 7 - Gathering Instruments; 8 - Evaluation Instructions; 9 - Evaluation Conduction; 10 - Data Packaging and; 11 - Evaluation Reporting.
Important to notice that the PEAR phases have to be executed, for each step, in that order. Finally, there are 32 activities that are distributed between phases and steps.

The Usa-DSL framework structure was planned in order to be adapted to the needs of each evaluation. It is possible to begin the “Planning” phase from any of the steps present in the Usa-DSL framework. For example, the evaluator can start the evaluation planning by the “P1 Define Evaluators Profiles” activity, or by the “P3 Define Data Type” activity. This will improve the framework flexibility, since it allows different evaluator to start the evaluation based on the activities that they feel more comfortable with, the ones that they already have some data, or even the activities that are easier to perform for a specific DSL. Besides, if the evaluator wants to perform a step in each of the PEAR phases, that also is possible, for example, it is possible to execute all activities from step “1 - Evaluators Profile” in all PEAR phases before starting activities in any other step. Furthermore, not all steps have to be performed. Some of them might not be executed, for example, the “4 - Empirical Study Method (SE)” step is only needed if the end user will be involved.

Figure 2 shows a high-level diagram of the order in which steps/activities in the PEAR phases can be executed.

4.2 Usa-DSL Phases

As mentioned before, the Usa-DSL framework contains the PEAR phases (see Figure 3). Each phase has a set of activities that is related to a respective step.

Phase 1 - Planning: in this phase, the evaluator organizes the planning of the aspects that will be used in order to evaluate the DSL. In this phase, documents must be defined and created, as well as decision-making about the data that has to be collected or what kind of user will be part of the evaluation, for example. To summarize, this phase is where the structure and planning of the evaluation will be constructed.

Phase 2 - Execution: in this phase, the documents created are used, subjects are recruited, environments are created and the evaluation is performed, following the already defined protocol.

Phase 3 - Analysis: this phase aims to accomplish the analysis of the artifacts created on the Planning and Execution phases. On the Planning phase, this analysis is executed in order for the documents to be adapted and, therefore, the decisions about the evaluation execution can be made. In this phase, the analysis is focused on the collected data and tasks created.

Phase 4 - Reporting: in this phase, the evaluator registers the used protocol, the created artifacts and analyzed data.

4.3 Usa-DSL Steps

The Usa-DSL framework is composed of eleven (11) steps. The steps of the Usa-DSL framework are described next (see Figure 3).

Step 1 - Evaluators Profiles: in this step the evaluator profile is defined, instruments to identify the evaluator are applied, the evaluator profile is analyzed and a report on that is written [1, 4, 11, 12, 16].

Step 2 - Ethical and Legal Responsibilities: similarly to the DECIDE Framework, which is an evaluation guide [25], Usa-DSL follows the best practices of ethical and legal issues to protect the user data, dignity, anonymity and well-being. Furthermore, it has to include some description to inform the users that they can stop the evaluation at any time they are not comfortable with some aspects of the evaluation process. At the end of this step, all the signed documents from the subjects are organized.

Step 3 - Data Type: in this step the type of data that will be used is defined, i.e. the evaluator defines whether the collected data is quantitative, qualitative or both. This will depend on the method that will be used, for example, usability testing uses quantitative, while user observation can use qualitative data. Basically, this step contains only one activity that is performed during the Planning phase.

Step 4 - Empirical Study Method (SE): the Empirical Study Method suggested for Usa-DSL is based on the Wohlin et
al. [37] proposal, which can be a survey, a case study or a controlled experiment. These methods can be defined based on, for example, the evaluator’s profile (Step 1) or the data that will be collected (Step 3). The Empirical Study Method can be used with other evaluation methods, e.g. usability testing or heuristic evaluation. However, the restrictions and characteristics of every method must be always respected.

Step 5 - Evaluation Method (HCI): the evaluation methods defined on Usa-DSL can be, for example, user observation evaluation, usability testing, inspection evaluation, or heuristic evaluation. The user observation evaluation must be applied when the study intention is to obtain the end users opinion about the DSL usability aspects. The inspection evaluation aims to verify the relevance of the language on the usability specialist level.

Step 6 - Metrics: the metrics used on Usa-DSL were defined from an SLR mapping [27]. They are comprehension/learning, ease of use, effort/conclusion time, observed complexity and efficiency. These metrics will guide the definition of the evaluation instruments questions to be applied during the evaluation. Similarly to Step 3, this step has only one activity performed during the Planning phase.

Step 7 - Gathering Instruments: the instruments were based on the studies of [25] and [28], e.g. heuristic checklist, ergonomic checklist, questionnaires, interview, use observation or user action recording.

Step 8 - Evaluation Instructions: according to Wohlin et al. [37], the evaluation instructions can be composed of use manual, instruments or task to be performed. These instruments must be distributed and used when executing an
empirical method. They are used, for example, to clarify the participants of the evaluation on what will be evaluated and when the evaluation will take place.

Step 9 - Evaluation Conduction: this is the step in which the aspects defined on the previous steps are applied. Therefore, it is necessary that the previous steps were executed and tested thoroughly, before involving the evaluation participants. Hence, a pilot test must be executed prior to the application of the evaluation to the actual participants. This will guarantee that the evaluation is viable. Furthermore, it is also important to guarantee that the needed number of participants will be achieved, otherwise, the results may not be statistically relevant for a quantitative evaluation.

Step 10 - Data Packaging: when the evaluation is finalized, the material used for training and the collected data should be stored in order to allow the study replication when necessary. This will allow future language evaluation and its comparison with the new collected data.

Step 11 - Evaluation Reporting: this report must follow the evaluation method that was chosen in step “5 - Evaluation Method (HCI)”. Each evaluation method provides a specific report with different fields that must be filled.

4.4 Usa-DSL Activities

The Usa-DSL framework activities are composed by a set of actions used to plan, execute, analyze and report the evaluation. The full set of activities can be seen in Figure 3. The description of the activities from all phases can also be found at https://github.com/Ildevana/Usa-DSL/wiki/Usa-DSL-Structure.

It is worth mentioning that the identification of each of the 32 activities is composed of an ID and its name. ID is composed of a letter and a number. The letter represents a phase and the number a step, e.g. “E5 Prepare the Evaluation” is an activity that belongs to phase Evaluation and is associated with the “5 - Evaluation Method (HCI)” step.

4.4.1 Planning Phase Activities

The Planning phase contains eleven (11) activities. These activities define the whole evaluation protocol. They are:

P1 - Define Evaluators Profiles: the goal of this activity is to define the evaluators profiles, which will be related to the evaluation method that will be used. The evaluation can be performed by, for example, an HCI expert, a domain analyst, a domain developer or a domain tester.

P2 - Define Informed Consent Term: it is a formal document that describes the evaluation goal, how the evaluation will take place, how the data will be collected, how the data will be protected, and so on. Usually, it is recommended the use of ethical codes from organizations like, for example, The Association for Computing Machinery (ACM).1

P3 - Define Data Type: the collected data type from the evaluation can be quantitative and/or qualitative. The quantitative data are numeric results that predict the quantity of answers attributed to determined item of a question. The qualitative data is composed of subjective information related to the participant’s opinion about the studied object. These data aim to predict what kind of information the evaluator intends to obtain. Albuquerque et al. [1] suggest the use of two data types, in order to obtain a wider and more complete view about the participant opinions. Barisic et al. [4], on the other hand, use quantitative data and consider that to be sufficient for the goal of their research.

P4 - Define Empirical Study Method: there are different empirical evaluation methods that can be used to evaluate usability. These methods have to involve users during data collection. This activity is closely related to activity P2. Examples of empirical methods are: Controlled Experiment, Survey or Case Study.

P5 - Define Evaluation Usability Type: as mentioned in the description of step “5 - Evaluation Method (HCI)”, evaluation can be through end users, HCI or DSL experts. This activity is related to activities P1, P3 and P4.

P6 - Define Metrics for Language Validation: the metrics depend on the evaluation goal and usability criteria that someone wants to evaluate. Examples of criteria that may be evaluated are: easy to learn, easy to remember, easy to use, effort/conclusion time, perceived complexity, utility, satisfaction, conclusion rate, task error rate, efficiency or effectiveness.

P7 - Define the Instruments of Data Gathering: some of the instruments that can be used to collect data can be heuristic checklist, log capture, use observation, interview or questionnaire.

P8 - Define the Instruments of Instruction and Training: the Usa-DSL framework use the following instruments: DSL guide, user scenario and language presentation. This activity also defines the tasks that will be executed by the user, when an empirical method is chosen. In that case, this activity has a close relation to P3, P4 and P5.

P9 - Define Execution Place: the place where the evaluation will take place depends on the data type that will be collected, the empirical study method that was chosen or even the usability type. For example, places could include a laboratory, via e-mail or through web, or even the users work place.

P10 - Define Data Storage: data packaging is an important activity, since this data might be used later in to replicate the evaluation.

P11 - Define Study Reporting: this activity is responsible for describing the way the results of the evaluation will be registered.

4.4.2 Execution Phase Activities

The Execution phase is composed by eight (8) activities. Each of these activities is used after the Planning phase step. They are:

E1 - Apply Instruments to Identify Profiles: questionnaire that characterizes the profile of the evaluation of participants is applied. This document is used to obtain information such as: DSL/Domain experience time, training, and area of activity.
E2 - Introduce the Form and Collect Signatures of Subjects: in this activity, the consent form must be presented to the participants and after their reading and consent, it must be signed and a copy is given to the researcher that is conducting the evaluation. The consent form provides the subjects with sufficient written information to decide whether to participate in a research study or not.

E4 - Develop and Conduct Protocol: this activity consists of developing the evaluation protocol, describing all the steps and documentation that will be used, such as the type of evaluation, experimental study, context, hypotheses, variables of the study, profile of the participants, the instruments, type of data, data storage and how the study will be reported. This protocol must be performed by the researcher carefully following the planned steps and activities.

E5 - Prepare the Evaluation: the evaluation instruments should be organized, the equipment arranged in the rooms, the participants must be available at the scheduled date and time, and the questionnaires answered.

E7 Data Collection: by applying the characterization questionnaires, collecting instruments and obtaining the data recorded in audio or video, they must be compiled and stored for later tabulation, transcription and analysis.

E8 - Introduce Instruments of Instruction and Conduct Training: the presentation and training are intended to guide the functioning of the language, regarding syntax and semantics, as well as to instruct on the usage scenario, that is, explain the task to be performed in the evaluation process. The delivery of the language manual and usage scenario refers to the delivery of the printed or online documents. These documents describe the functioning of the language and its syntax and semantics. They contain the description of the usage scenario that must be expressed as a requirement or task.

E9 - Execution of Tasks and Evaluation Conduction: the task must be modeled according to the usage scenario delivered to the participants and must be performed from the tool that supports the execution of the language. Upon completion of the task modeling, the researcher may conduct an interview with the participants and thereby obtain their opinion about the language being evaluated. In addition to the interview, the researcher can choose only the use of the questionnaire, filled by participants after completing their tasks.

E10 - Store Data Obtained: after performing the evaluation, the collected data should be stored in a database or other location in order to compile the data later. If data are quantitative, it is important to tabulate them so that their behavior can be observed later and thus to obtain conclusions. If the data is qualitative, it is important to process the interviews, annotations, answers to open questions, recordings and access logs, trying to obtain patterns and a set of relevant information for the study.

4.4.3 Analysis Phase Activities
As mentioned before, the Analysis phase contains five (5) activities:

A1 - Analyze Evaluators Profiles: the analysis of the profiles is used to gather the number of participants and the type of knowledge they have. These profiles can be classified as: Beginner - one who does not have solid knowledge on the domain or on DSL. Intermediate - one who has some knowledge on the domain and/or on DSL Advanced - one who has solid knowledge on the domain and on DSL.

A4 - Analyze the Developed Protocol: in the analysis activity of the study protocol, all the described steps should be reviewed in detail and how they will be performed in order to ensure the validity of the study.

A7 - Analyze the Collected Data: when analyzing the data collected during the evaluation, standardization, hypothesis testing, analysis of images and logs, transcription of interviews and videos are performed.

A9 - Analyze the Performed Tasks: the developed models should be checked by more than one researcher to verify and to obtain the task execution rate and the error rate performed by the participants. After, the evaluation of those that did not reach the objectives of the task, or did not complete the intended task, will be discarded.

A11 - Analyze the Documentation: the documentation used in the evaluation must all be analyzed by the researcher and checked by a second researcher to ensure the consistency of the produced information and documentation.

4.4.4 Reporting Phase Activities
The final phase is Reporting, which is composed of eight (8) activities that aim to register what was performed during the previous evaluation phases. These activities are:

R1 - Report Evaluator Profiles: when reporting the participants’ profile, the classification and the total number of participants who performed the evaluation should be taken into account. Furthermore, other information should be described if it appears in the characterization questionnaire.

R2 - Report Subjects Number and the Form Used: all documents used in the evaluation should be described in detail and attached to the final report.

R4 - Report the Developed Protocol: the study protocol should be described for each planned, executed and analyzed step.

R3 - Report Conduction Evaluation: HCI evaluation methods must be described: Usability Testing - this evaluation aims to test whether potential users would use the language developed to model the domain to which it was proposed; Heuristic evaluation - this is a usability inspection method, which is applied by HCI specialists who are guided by a set of heuristics developed by Nielsen. This method aims to identify usability problems in the evaluated language.

R7 - Report Data Analysis: Quantitative data - should be reported through charts, spreadsheets or hypothesis testing. Qualitative data - can be represented by an image, interview transcript, annotation excerpts, categorization and standards, high-level video narratives, and fragments of open-ended questions.

R8 - Report the Instruments: the instruments used, charac-
terization questionnaire, language manual, usage scenario, interview script, opinion questionnaire, among others, must be detailed at high level in the protocol and arranged in an appendix in the document used to present the study.

R9 - Report Tasks Analysis: the evaluation must be reported according to the chosen method. The usability test will be reported through the protocol of an experiment, case study or survey. When a heuristic evaluation is performed, the analysis performed by the specialists, as well as the activities carried out and the generated models, should all be reported.

R11 - Report the Results and Analyzed Information: at the end of the evaluation the data should be fully described in a report format, containing all the documents attached to the report.

5. EVALUATION: FOCUS GROUP

In order to evaluate the Usa-DSL framework several strategies could have been used, for example, a focus group [20] or an empirical controlled experiment [37]. This paper presents the evaluation performed using a Focus Group method, which gathers qualitative data during group discussion sessions. This method was chosen because it is a useful method that can be used to measure the reaction of specialists and, therefore, some straightforward conclusions can be drawn by the group. A focus group is organized in phases [9, 18, 20]: planning, preparation, moderation, and data analysis and reporting (see Figure 4). Usually, a focus group is composed by a moderation team (normally an interviewer/facilitator and moderation assistants) and a set of subjects. For the evaluation subjects, HCI, Software Engineering and Performance Testing experts were invited.

![Figure 4: Focus Group Process](http://porteiras.s.unipampa.edu.br/pampatec/)

Furthermore, to verify whether the focus group phases were ready to be applied to the subjects, a test pilot with two subjects was used. These subjects belonged to the DSL Canopus project [7] [8]: a project analyst and a developer. After the test pilot, some modifications were applied to the framework structure before submitting it to the focus group, for example, timings were altered, the questionnaires glossary was improved, and the number of activities that would be discussed was reduced. These two subjects were later involved in the focus group as assistants, one as recorder and other as timekeeper.

The next sections detail the results from each phase of the focus group.

5.1 Planning and Preparation

The main goal of the focus group was to validate the Usa-DSL framework in order to understand whether the framework phases, steps and activities would effectively prepare a usability protocol to evaluate the usability of a DSL. Hence, the planning and preparation phases had to allow the subjects, during the moderation phase, to understand the framework structure and objectives. Furthermore, the planning and preparation had to be able to produce good discussions among the subjects during the moderation phase. Hence, the discussion session guide, documents to be presented to the subjects, questions that needed answers, and all the environment for the focus group, were planned/ prepared during these phases. All these were previously verified in the test pilot, as mentioned before.

During the planning phase, the goals of the focus group, the profiles of the subjects, the way the discussion would be conducted, the role of the interviewer and the assistants, date and place for the focus group, and which documents would be used, were defined. The goal was already mentioned in the previous paragraph. Date and place were set as 2017, May and a technological park from a federal university\(^2\), respectively. Also, as mentioned before, the subjects had to have experience on using or designing DSLs, or understanding of HCI evaluation. Some of the subjects had knowledge on both HCI and DSLs. In the end, seven subjects were selected to be part of the focus group.

In order to prepare the environment, and to avoid any kind of interruption during the discussion session, the following preparation was executed prior to the evaluation of the Usa-DSL framework: 1) the meeting room was prepared with some audio and video recording equipment; 2) all the printed documents were reviewed and accounted for; 3) audio and recording equipment were tested.

In order to assist the subjects to visualize the framework structure a board with the Usa-DSL framework (see Figure 3) was always available for the subjects. Subjects also used post-it, pens, and had access to the guide, informed consent term and questionnaire.

5.2 Moderation

In the moderation phase, it was ensured that the subjects felt comfortable, respected and free to expose their opinions [18]. To achieve this goal, a script was followed as a guide. This script presents a welcome message to the subjects, the instructions on the “Informed Consent Term”, the completion of the profile questionnaire and the printed documents that would be used during the session\(^3\).

In order to achieve this goal, a guide as a script was suggested. This script shows a welcome message to the subjects, the instructions about the Informed Consent Term, the completion of the characterization questionnaire and a document describing all activities that would be performed during the session.

To mitigate understanding problems during the discussion

\(^2\)Technological Park of Pampa (PampaTec) from UNIPAMPA. http://porteiras.s.unipampa.edu.br/pampatec/

\(^3\)Documents are available at http://tiny.cc/SAC-UE-2018.
During the session parts, the mediator would allow free discussions among the subjects. The mediator would only intervene when the discussion would get out of the scope of the goal of the focus group, or when some of the subjects was not participating in the discussion. The subjects were free to discuss any topic (in each part of the session, as explained before), and they would decide what had to be performed regarding each topic, i.e. to maintain, to join, to modify, to include, to change syntax, to change semantics, or to remove something. In each part of the session, the group would elect a rapporteur that was responsible to fill the questionnaire at the end of each part of the discussion session.

The duration of the discussion session was two hours and twenty minutes (2h20min), including the presentation time of each topic, its objectives and intentions. The opinions expressed by the subjects were recorded in audio and video and later transcribed. Namely, the audio recording was used to support the video recording, so that it could help in understanding the discussion.

### 5.3 Data Analysis

The phase of analysis and interpretation of the generated data constitutes an important part of the qualitative research, considering the context, the behavior and the perception of the subjects [18] [20]. For the data analysis phase, the audio from the video was transcribed and the recorded audio was used when the sound of the video was not clear. The transcript followed the order in which the study script was planned, separating the discussion by session and comparing with what was reported in the questionnaire delivered by the rapporteur. The analysis presented in this section were firstly performed by one researcher, and later the conclusions were discussed with a second researcher to validate the results. The subjects experience, expressed in the profile questionnaire and presented in the Table 2, was also taken into account.

The next sections present a summary of the subjects discussions. This summary was based on the transcription of the recordings performed during the discussion session.

#### 5.3.1 First Session Part: Usa-DSL Steps

At the start of this session part, the subject identified by S1 mentioned that he had already read the material that he had received by e-mail. Furthermore, he also said that the framework seems to be for generic enough to be used not only for DSLs. He questioned the reason for the existence of steps "3 - Data Type" and "4 - Empirical study Method", but as the discussion progressed he understood that one step is related to SE and the other one to HCI. S2 mentioned the "3 - Data Type" step and asks the other subjects why the "3 - Data Type" step should be defined before the "4 - Empirical Study Method" step. He mentions that if he knew how to use a particular method, it would be easier to define the "3 - Data Type". However, S5 said that the order of the steps "3 - Data Type" and "4 - Empirical Study Method" makes sense and argued with S2 that it would not be enough to be an expert in a method to perform an evaluation. In the first interaction of subject S4, he expressed his idea that steps "3 - Data Type" and "4 - Empirical Study Method" should be merged, but was convinced by the explanation of S6 that the data type must be defined before the empirical study method and stated that the data to be collected can change the method to be applied. S3 agreed that "3 - Data Type" must be defined before the "4 - Empirical Study Method"; at this point the interviewer instigated S7 to participate in the discussion, but he did not have anything further to add.

"[...] the order of the Data Type and Empirical Study Method makes sense, if I am, for example, an expert in an empirical study method it would not be enough to carry out an evaluation, because
if the data we want to obtain is quantitative and he only knows how to do Case Study this would not solve [...]” (S5)

“[...] the data to be collected can change the empirical study method to be applied.” (S6)

The second issue raised was regarding step “6 - Metrics”. Subjects S2 and S4 repeatedly questioned whether metrics should be selected from the data type or whether they would depend on the empirical study method and whether metrics could be changed or new metrics could be included at the time of the evaluation. After a lot of questioning, S4 concluded that the choice of metrics is based on the data type and the group was persuaded to maintain the order.

The last issue to be discussed was on step “8 - Evaluation Instructions”. S7 suggested that it should be placed before step “6 - Metrics”, but quickly S5 replied that it was not possible to instruct someone about the evaluation before beginning the evaluation. After that, this issue was considered as resolved by the subjects.

At the end of the topic, when the rapporteur began to respond the questionnaire, the group suggested reading item by item so that, in common agreement, the alternatives, justifications and changes would be described. Basically, the subjects strongly agreed with most questions that were asked in the questionnaire. Although they believed step “7 - Gathering Instruments” could be changed to “7 - Evaluation Tools”, in the end, they did not really suggested that change since it was not mandatory. After 15 minutes of discussion, the group decided not to modify the steps of the framework.

“[...] cannot instruct on the system operation before preparing the evaluation.” (S5)

This section described how the subjects behaved during the discussion session. The next sections do not present the way they discussed, but describe a summary of the discussion in each session part.

5.3.2 Second Session Part: Usa-DSL Phases

Initially, Usa-DSL was composed of the following phases: “Definition”, “Execution”, “Analysis” and “Results”. However, during the discussion session it was clear that the “Definition” term should be wider, and, therefore, it was changed to “Planning”. There were some questioning related to the “Execution” and “Results” terms. The subjects were not convinced when the evaluation data collection, recording and results dissemination activities should be performed. In the end, there was a general understanding that the “Execution” phase should include activities related to data collection. Besides, the “Results” phase name was changed to “Reporting”, since it includes the activities “Record of Results” and “Data Collection”.

“[...] the term “Results” does not seem to be a good name for a phase [...] results gives a discontinuity perception, and it seems that the evaluation finishes there and that there is nothing else to be done [...]” (S4)

5.3.3 Third Session Part: Usa-DSL Activities

At the beginning of the third session part, a board with the complete view of the framework was presented to all the subjects. This board contained the framework phases, steps and activities, even though during this part of the session the goal was to discuss only the framework activities (see Figure 3). Furthermore, a document containing each activity description was available. First, the description was read and discussed by the subjects.

First, there was a discussion on the activities names and whether they were included in the right phase. There was not questioning regarding the step in which the activity was included to. After that, the subjects chose an activity randomly to start the discussion. Some subjects questioned the importance to include the place in which the evaluation would take place, and also if this could be included in an activity called “Define and Conduct the Evaluation”. In the end, the subjects considered that it would be important to keep the activity as “Define Execution Place”.

In order to organize the discussions, the subjects decided to discuss the activities by phase. There were some suggestions to split activities (“Prepare and Conduct the Evaluation”), to join activities (“Execution of Tasks” and “Evaluation Conduction”) and to create new activities (“Compile and Protocol Review”). Regarding this suggestions, S3 led the discussion and pointed out that their goal was to evaluate the Usa-DSL framework structure and not to describe activities following some usability evaluation method or software testing technique.

At the end of this part of the session, the subjects read the description of the activities again and decided to make some corrections on duplicated information in the activities. The subjects also considered that the examples mentioned in the “Define Experimental Study Method” activity could induce to someone to choose certain evaluation method. Hence, if that was not the intention, then that should be avoided.

During the discussions about the framework activities, there was a better understanding on the Usa-DSL framework. See some statements from some of the subjects:

“The execution of the framework works as a matrix that crosses steps and phases that results in activities.” (S4)

“The framework is a set of good practices.” (S6)

5.3.4 Fourth Session Part: Usa-DSL Structure

The last part of the discussion session was used to close the discussion and also to confirm the framework structure that was suggested by the focus group. Even though the duration of the discussion session was long, i.e. 2h20min, the subjects remained interested and engaged in the discussions. They confirmed that the structure of the framework was good and only some minor changes should be made, for example, change some terms names. Some minor com-
ments were added at this moment. For example, S4 suggested that a document describing recommendations or usage rules should be added. S1 mentioned that it seemed that an evaluator with no experience in usability evaluation would be able to carry out an evaluation using this framework. Two subjects, S1 and S6 said that it would also be important to have a workflow for an evaluator to make it easier for someone to follow the framework. The subjects also reported that they had plenty of time to discuss all the topics that were supposed to be discussed.

5.3.5 Usa-DSL Changes after the Focus Group

Some changes were proposed by the Focus Group and were incorporated in the final Usa-DSL framework (see Section 4). Those proposals were related to the name of the phases or inclusion of new activities. For example, subjects from the Focus Group suggested to use the term “Planning” for the first phase, rather than the term “Definition”, which was used before the Focus Group. Another name that the subjects suggested changing was the last phase name, from “Review” to “Results Registering”, which in the end was changed to “Reporting”. Regarding the inclusion of new activities, the subjects suggested the inclusion of two activities in the Planning phase, i.e. “Define the Instruments of Instruction and Training” and “Compile and Review Protocol”. While the former was included in the framework, the latter was not since it should be a sub-activity in all other phases. The last change suggested by the subjects was to gather together activities “Execution of Tasks” and “Evaluation Conduction”, which were two separate activities in the Execution phase. These two activities became activity “E9 - Execution of Tasks and Evaluation Conduction”.

It is worth mentioning that another suggestion that was incorporated to the final version of Usa-DSL was to include an identification to each activity, since this would facilitate future discussions and framework organization. Therefore, each activity contains an ID and its name, where the ID is composed by a letter and a number. This was also applied to the steps of the framework.

6. CONCLUSION

Domain engineers aim to, through the development of different languages, facilitate the creation of new concepts and theories in order to minimize the difficulties inherited from applications development. One way of minimizing this difficulties is to use Domain Specific Languages, DSL. Although these languages help the developers, their usability has to be analyzed in a thorough way.

This paper presented a framework that will help DSL developers to evaluate the usability of the languages that they are proposing. This framework was evaluated using a focus group method, which confirmed that the framework will help DSL developers. The subjects that participated in the Focus Group had previous experience developing or using DSLs and they believe their job would have been easier if they had the framework to help to improve the DSLs.

For future research, it would be important: 1) To accomplish the evaluation of several DSLs, preferably using different usability methods and also the Usa-DSL framework; 2) To evaluate DSLs using different evaluator profiles or different evaluation frameworks; 3) To propose a process in order to assist the activities presented in the Usa-DSL framework; 4) To improve artifacts, such as: checklist, manuals, questionnaires and protocols, which will support the evaluation process and the Usa-DSL framework; 5) To evaluate empirically the artifacts developed using the Usa-DSL framework.

7. ACKNOWLEDGEMENT

The authors would like to acknowledge DELL, CNPq and CAPES for their financial support in the development of this work.

8. REFERENCES


ABOUT THE AUTHORS:

Ildevana Poltronieri is a PhD student in Computer Science. She received her Master in Computer Science from Pontifical Catholic University of Rio Grande do Sul – PUCRS in 2018, her Bachelor in Information Systems from PUCRS in 2016, and a degree in Mathematics from URCAMP in 2003. She also holds a specialization in Applied Mathematics in 2006. She has experience in the area of Computer Science, with emphasis on Information Systems. She is currently a researcher collaborator in the Laboratory of Empirical Studies in Software Engineering - LESSE UNIPAMPA. She researches in the fields of Human-Computer Interaction and Software Engineering. Her research interests include Software Product Lines, Domain-Specific Language, and Usability Evaluation.

Avelino Francisco Zorzo is a full time professor at Pontifical Catholic University of Rio Grande do Sul (PUCRS), coordinator for professional post-graduate accreditation for Ministry of Education in Brazil, and Member of Brazilian Computer Society. He received his Computing Science PhD from Newcastle University in England in 1999, and a post doctorate at the Cybercrime and Computer Security Centre at Newcastle University in 2012. He was Industry Liaison and Education Director at Brazilian Computing Society, head of Computing Science School at PUCRS, member of Ethical in Research Committee at PUCRS, director at IT Users Society in Rio Grande do Sul, and member of Board for the Association of IT Companies in Rio Grande do Sul. His main research interests are security, dependability, and operating systems. He has published more than 100 papers in refereed conferences and journals.

Maicon Bernardino received his PhD in Computer Science at PUCRS in 2016. He worked for the CST in Analysis and Systems Development at FTEC, the CST in Digital Games at FEEVALE, and Federal University of Technology - Paraná (UTFPR). Currently, he works as an adjunct professor at Federal University of Pampa (UNIPAMPA) and the coordinator of Laboratory of Empirical Studies in Software Engineering (LESSE). His work experience involves Project Management, Performance Testing, Model-Based Testing, DSL, SPL, Gaming, Databases, and Software Testing.

Marcia de Borba Campos received her PhD in Informatics in Education from Federal University of Rio Grande do Sul in 2001. She is a CNPq researcher of the Brazilian Computer Society, and a member of the UX group from Association of IT Companies in Rio Grande do Sul. She worked for Pontifical Catholic University of Rio Grande do Sul as an adjunct professor (1996-2017). She was a member of the Ethical in Research Committee at PUCRS, an Internship coordinator at PUCRS, an Educational Technologies coordinator at PUCRS. Also, she worked as an adjunct coordinator of Virtual and a coordinator of assistive technologies in the Faculty of Computer Science at PUCRS. Her main research interests are Human-Computer Interaction, Usability, UX Design, Assistive Technologies, and Informatics in Education and Games. She has published more than 80 papers in refereed conferences and journals.
Adaptive Information Distribution for Dynamic Sets using Multicast Push and Pull

Matthias Prellwitz
Institute of Computer Science
University of Rostock
Rostock, Germany
matthias.prellwitz@uni-rostock.de

Helge Parzyjegla
Institute of Computer Science
University of Rostock
Rostock, Germany
helge.parzyjegla@uni-rostock.de

Gero Mühl
Institute of Computer Science
University of Rostock
Rostock, Germany
gero.muehl@uni-rostock.de

ABSTRACT
When developing applications in the area of ubiquitous computing, the developer often does not know at design time which and how many devices will be available at runtime when the user deploys the application in his personal environment. To tackle this problem, we proposed a new programming abstraction called dynamic sets in previous work. A dynamic set allows an application to interact transparently with a set of remote objects in the same way it interacts with a single object. The developer programs against a proxy that replicates the calls to the objects and aggregates the return values. While dynamic sets make application development easier and more flexible, efficiency problems can arise when an application is calling proxy methods with high frequency as if the object was local (since the calls are replicated to all members of the dynamic set).

In this paper, we propose adaptive heuristics that tackle this problem for getter methods returning information about the objects’ state. The heuristics use unicast and multicast communication and dynamically switch between pull (application requests update from object) and push (object sends update to application) depending on the getter rates of applications and the setter rates of objects to optimize communication costs. After describing the basic interaction patterns and presenting the details of our heuristics, we present the results of a comprehensive evaluation based on discrete event simulations. The results show that our heuristics effectively reduce communication costs although they operate on the application layer and, thus, have no detailed knowledge about the physical network topology.

CCS Concepts

• Software and its engineering → Distributed systems organizing principles; Client-server architectures; • General and reference → Metrics; Evaluation; Performance;

Keywords

Push/Pull, Heuristics, Adaptivity

1. INTRODUCTION
Ubiquitous computing and the Internet of Things (IoT) are becoming an important part of our daily lives. Mobile devices such as smartphones and tablets can form intelligent ensembles with sensors, actuators, and processors embedded in smart home devices such as TVs, projectors, audio systems, central heating, air conditioning, and window shades. This way, mobile devices can be used for visualizing and controlling our environment making our lives more comfortable and productive. However, while the opportunities are constantly increasing, developing and maintaining applications for intelligent ensembles is still a challenging and cumbersome task. One major reason for this is heterogeneity with respect to programming interfaces, interaction paradigms, and transport protocols. Another major problem is that the application developer often does not know at design time which and how many devices will actually be available when the user deploys the application in his personal environment. While the former problem can be tackled with traditional middleware approaches, new approaches are needed to solve the latter problem.

To resolve the gap between design time and deployment time, we propose a new programming abstraction called dynamic sets. A dynamic set allows an application to interact transparently with a variable number of remote objects having the same interface in the same way as it interacts with a single of these objects. With our approach, remote objects can be application objects or they can represent sensors and actuators interacting with the environment. Which object belongs to a dynamic set, is defined using code annotations in the application source code, via an optional deployment descriptor, or programmatically using the dynamic set API.

To bundle a variable number of objects, the application developer programs against a proxy. This proxy hides the access to the remote objects, maintains object selection criteria, replicates function calls, and aggregates return values. Fig. 1 depicts the concept of dynamic sets that was introduced in previous work [8, 9]. In the basic setting, the application does not need to be aware that it is interacting with multiple objects since it only calls methods of a single local proxy. In advanced settings, in which the application is aware of the dynamic set, it is possible to enrich the proxy’s interface with new methods for object sets, e.g., subsets, in-
have presented in a previous paper [10] and derive improved
In particular, we revise three adaptive heuristics that we
will be prevalent for which pair of application and object
where the other approach is inefficient. Because it is in gen-
dual in the sense that they are efficient in those scenarios
ers. Moreover, it became obvious that push and pull are
suitable for certain scenarios, while it is inefficient for oth-
This argumentation shows that each of the two approaches is
been delivered to the application.
the respective objects) overwrite information that has not
application requests new information with a much lower fre-
a too high frequency. This is especially inefficient when the
push-based approach works well for methods only returning
state information, it can obviously not simply be applied to
methods that change the state of the objects. Moreover,
push can also cause a performance problem when (at least
some of) the objects are pushing information updates with
a too high frequency. This can not only lead to network
congestion, but might also overload the nodes hosting the
objects. For methods that only return state information,
this polling-based approach is especially inefficient when the
application calls a method with a much higher frequency
than the requested information actually changes. Then, the
majority of calls do not return any new information.
In such a scenario, it would be beneficial if, instead of let-
ting the application poll for information updates, the objects
were pushing available updates to the proxy. The proxy,
then, caches the newest information for each object so that
the most up-to-date information can be returned immedi-
ately when the application requests it. However, while this
push-based approach works well for methods only returning
state information, it can obviously not simply be applied to
methods that change the state of the objects. Moreover,
push can also cause a performance problem when (at least
some of) the objects are pushing information updates with
a too high frequency. This is especially inefficient when the
application requests new information with a much higher fre-
quency. In that case, most of the information updates (from
the respective objects) overwrite information that has not
been delivered to the application.
This argumentation shows that each of the two approaches is
suitable for certain scenarios, while it is inefficient for oth-
ers. Moreover, it became obvious that push and pull are
dual in the sense that they are efficient in those scenarios
where the other approach is inefficient. Because it is in gen-
eral difficult to decide at design time which kind of scenario
will be prevalent for which pair of application and object
at runtime, we investigate adaptive approaches that switch
between push and pull at runtime.
In particular, we revise three adaptive heuristics that we
have presented in a previous paper [10] and derive improved
variants that better exploit multicast communication. The
first heuristic decides whether unicast push or unicast pull is
used for each pair of proxy and object. The second heuristic
decides for each object, whether it pushes its information up-
dates to all applications for that the object is a member of its
dynamic set using multicast or all applications have to pull
for information updates individually. The third heuristic
uses multicast to push information updates to those appli-
cations having the highest request rates, while applications
with lower request rates have to pull individually. Although
these basic heuristics partially employ multicast communi-
cation for pushing data updates, they all are limited to in-
dividual unicast requests if new information is pulled. The
novel heuristics, however, leverage multicast for both push
updates and pull requests and, thus, further improve the
performance. After describing our heuristics and their new
variants, we thoroughly evaluate and compare them using
discrete event simulations. The results show that our ap-
proaches indeed reduce the network consumption.
The remainder of the paper is structured as follows: Sect. 2
explains the basic pull-based and push-based interactions
used by our adaptive approach and Sect. 3 introduces our
heuristics and their variants. Then, we present an evaluation
for different settings in Sect. 4. The paper closes with related
work in Sect. 5 and our conclusions in Sect. 6.

2. INTERACTION PATTERNS

In this section, we describe the basic interaction patterns
applied by our heuristics as well as the corresponding cost
model used. The interaction patterns can be categorized by
the fact whether they use pull or push and whether they
use unicast or multicast. We consider unicast pull, unicast
push, multicast push, and multicast pull. The cost model
enables proxies and objects of dynamic sets that interact
with each other to estimate the communication costs and,
thus, to derive adequate adaptation decisions.

We discuss the interaction patterns in the following four sub-
sections. Thereafter, we explain our middleware architec-
ture. For that, we introduce the following notations: The
set $A$ denotes all applications $\{a_1, \ldots, a_i\}$ deployed, whereas
the set $D$ contains all devices $\{d_1, \ldots, d_j\}$ available. Fur-
thermore, each application $a$ has an associated dynamic set
$S_a$ containing a subset of the devices. Finally, for each de-
vice $d$, $A_d = \{a \mid d \in S_a\}$ is the set of all applications for
that $d$ is in the application’s dynamic set $S_a$.

2.1 Unicast Pull

If an application $a$ invokes a method on a dynamic set $S_a$,
the proxy of the dynamic set takes care that the invocation
request is replicated and forwarded (usually in parallel) to
the devices that are currently member of the set. At each
of those devices, the method is invoked and the return value
is sent back to the proxy, where our middleware aggregates
the return values to a final result. Hence, invoking a method
on a dynamic set causes two messages to be sent for each
set member: one message for the request and one for the
corresponding reply.

We define the communication costs $C_{UC}^{pull}$ to equal the num-
ber of messages caused by unicast pull over an observed time

Figure 1: Basic idea of dynamic sets.
period. It depends on the set size $|S_a|$ and on the number of
method invocation requests $r_a$ in the observed time period:

$$C_{MC}^{Pull} = 2 \cdot r_a \cdot |S_a|$$  \hspace{1cm} (1)

### 2.2 Unicast Push

For many applications, up-to-date information is important and,
therefore, these applications usually pull devices for new data frequently. In such situations, an alternative approach is to push
new data from devices to interested applications as soon as the data is available. Without an explicit request, the device $d$ sends
a message with the new data to all applications in $A_d$.

Hence, communication costs for unicast push $C_{UC}^{Push}$ equal
the number of messages sent by device $d$ to applications in $A_d$. The costs are, thus, proportional to both the size of $A_d$
and the number of information updates $u_d$ that occurred at
device $d$ during the observed time period. For unicast push, the number of messages are, thus, calculated as follows:

$$C_{UC}^{Push} = u_d \cdot |A_d|$$  \hspace{1cm} (2)

### 2.3 Multicast Push

With unicast push, the device sends a message with the
information update to each application individually. Leveraging
unicast communication, the device can send an update
to all applications simultaneously provided that the
network infrastructure supports multicast communication
e.g., IP multicast) and the multicast groups are adequately
configured. Then, the network elements (e.g., routers and
switches) take automatically care and replicate a message
where necessary, i.e., where delivery paths to the receivers
split. It is easy to see that a multicast message to $n$
receivers is more expensive than just a single unicast message, but cheaper than $n$ individual unicast messages.

However, the exact costs are hard to determine as they
strongly depend on both the network topology and the
location of the receivers therein. The communication
overhead of a multicast message for $n$ receivers with $f(n)$
for the remainder of this paper, where $f(n)$ is a function
of $n$ such as $\log n$ or $\sqrt{n}$. Thus, the communication costs
for pushing $u_a$ information updates from a device $d$ to $|A_d|$
application receivers using multicast push are given by:

$$C_{MC}^{Push} = u_a \cdot f(|A_d|)$$  \hspace{1cm} (3)

### 2.4 Multicast Pull

Several middleware implementations [14, 6] offer to exploit
multicast communication to efficiently deliver a request to
multiple receivers. Leveraging such a request scheme, the
proxy of a dynamic set can, thus, forward a method
invocation to all devices of the set using one multicast message
only. However, each invoked device has to individually sent
its result back to the proxy using unicast communication.
Hence, multicast pull reduces costs for the request while the
reply remains unchanged when compared to unicast pull.

Estimating the costs of a multicast message to $n$
receivers with $f(n)$ as above, the communication costs $C_{MC}^{Pull}$
for requesting and pulling $r_a$ information updates from $|S_a|$ de-

### 2.5 Middleware Architecture

The middleware architecture of our approach is shown in
Fig. 2. On the left side, the proxy of a dynamic set of an
application is depicted and on the right side, the proxy of
one of the devices that are member of the dynamic set is
shown. For simplicity, we concentrate on getter and setter
methods. If the application invokes a set method, a request
is sent to the device and the resulting reply is passed back
to the application proxy. If the application invokes a get
method and the communication is in pull mode, the same
request/reply interaction occurs. If multiple devices are in
the application's dynamic set, request/reply communication
takes place for each of these devices. If the communication
is in push mode, the value cached in the application proxy is
instantly passed back to the application. And, if an update
occurs at the device, a notify message is sent to the applica-
tion proxy where the cached value is updated accordingly.
If a request or an information update is sent to multiple
receivers (i.e., devices or applications, respectively), either
individual unicast messages or a single multicast message is
used. Whether communication is switched from pull to push
or vice versa and whether unicast or multicast is used, is
decided based on gathered statistics stored at the application
proxies and the device proxies. Additionally, we also con-
sider to push information updates only to a subset of applica-
tion proxies using multicast, while the remaining proxies
are pulling information updates.

### 3. ADAPTATION HEURISTICS

Our middleware architecture allows to switch the communi-
cation paradigm (e.g., push/pull, unicast/multicast) at runtime whenever it seems beneficial, i.e., if we expect to save
a significant number of network messages. In the following,
we present three adaptation heuristics that each leverage dif-
f erent communication patterns in order to reduce network
communication costs.

#### 3.1 Link-based Heuristic

The first heuristic $h_{link}$ considers each link individually. In
particular, a link $(a, d)$ between an application $a$ and a de-
vice $d$ can be in one of two modes: either the application pulls information from the device via unicast pull requests or the device pushes information to the application via unicast push updates. The heuristic uses Eq. 1 and 2 to compare the costs of pull and push, respectively, and switches the link into the mode that is expected to raise lower costs. Thus, $h_{\text{ink}}$ employs unicast pull, if
\[ C_{\text{UC}}^{\text{pull}} + T = 2 \cdot r_a + T < u_d = C_{\text{UC}}^{\text{push}}, \] (5)
and unicast push, if
\[ C_{\text{UC}}^{\text{push}} + T = u_d + T < 2 \cdot r_a = C_{\text{UC}}^{\text{pull}}, \] (6)
where $T$ is a configurable threshold of a hysteresis that counters oscillations between both modes.

To calculate and compare costs, the heuristic needs to monitor both the information request rate (called getter rate) $r_a$ of the application and the information update frequency (called setter rate) $u_d$ of the device (i.e., the number of requests and updates in the recent monitoring period, respectively). Since in pull mode, data updates are not automatically forwarded to the application side, monitoring the setter rate has to be done at the device. In push mode, however, monitoring the getter rate needs to be carried out on the application side as the requests are not forwarded to the device. Hence, the $h_{\text{ink}}$ heuristic requires a distributed implementation and evaluation of the decision criteria given above. This way, the device may switch the link into push mode, whereas the application can revert it to pull again.

### 3.2 Device-based Heuristic

When pushing data updates to multiple applications, the usage of multicast communication is advantageous. When compared to unicast, multicast communication is usually the more efficient the more receivers can be reached by a single message. In this context, our device-based heuristic $h_{\text{dev}}$ uses multicast instead of unicast to deliver its information updates. Based on the cost functions given by Eq. 1 and 2, the heuristic switches to multicast communication for all applications (i.e., for all dynamic sets in which the device is a member) if this is expected to reduce the overall network traffic. Particularly, $h_{\text{dev}}$ uses unicast pull for all applications/dynamic sets $A_d$ in which the device is member, if
\[ \sum_{a \in A_d} C_{\text{UC}}^{\text{pull}} + T = 2 \sum_{a \in A_d} r_a + T < u_d \cdot f(|A_d|) = C_{\text{MC}}^{\text{push}}. \] (7)
and multicast push, if
\[ C_{\text{MC}}^{\text{push}} + T = u_d \cdot f(|A_d|) + T < 2 \sum_{a \in A_d} r_a = \sum_{a \in A_d} C_{\text{UC}}^{\text{pull}}. \] (8)

Basically, the $h_{\text{dev}}$ heuristic sums up the request rates of all dynamic sets in which the device is a member and compares this sum to the costs of a multicast dissemination of the information updates. It is, thus, sensible to evaluate the decision criteria on the device side, but this requires the applications’ request statistics to be periodically transmitted to the device while multicast push communication is used. The threshold $T$ determines the size of the hysteresis that counters oscillations between both modes.

### 3.3 Hybrid Heuristic

If request rates differ to a large extend, it is not sensible to communicate with all applications in the same way, i.e., to actively push information updates either to none or to all of them. Although multicast push may reduce the overall network costs, for some applications, however, pushing information updates may cause a substantial overhead, especially if the setter rate is much higher than the application’s getter rate. To address this aspect, our device-based, hybrid heuristic $h_{\text{hyb}}$ does not switch between request-driven unicast pull and update-driven multicast push. Instead, it uses both modes simultaneously and classifies each application in order to assign it either to the pull group or to the push group whichever is expected to be most favorable for the given request/update pattern. By periodically readjusting the group assignment, the heuristic is able to dynamically adapt to changing traffic patterns. Since the pull or push group may also be empty, the $h_{\text{hyb}}$ heuristic includes both modes of the previous $h_{\text{dev}}$ heuristic as corner cases, where information updates are either pushed to none or all applications. In addition, $h_{\text{hyb}}$ enables fine-grained, intermediate configurations that lie between those two extremes.

For the $h_{\text{hyb}}$ heuristic, the key challenge is finding the group assignment that suits the request and update pattern best. Therefore, the set of applications $A_d$, in whose dynamic sets device $d$ is member, is split into two disjoint groups $R_d$ and $U_d$ that use request-driven unicast pull and update-driven multicast push, respectively. While the respective network costs can be estimated according to Eq. 1 and 3, the heuristic varies the group assignment in order to minimize the overall costs, i.e.,
\[ \min \left\{ C_{\text{UC}}^{\text{pull}}(R_d) + C_{\text{MC}}^{\text{push}}(U_d) \mid R_d \cup U_d = A_d, R_d \cap U_d = \emptyset \right\} \] (9)
where
\[ C_{\text{UC}}^{\text{pull}}(R_d) + C_{\text{MC}}^{\text{push}}(U_d) = 2 \sum_{a \in R_d} r_a + u_d \cdot f(|U_d|). \] (10)

Algorithm 1 solves this optimization problem. The procedure $\text{PARTITION}()$ gets the set of applications $A_d$ as well as the expected number of data updates $u_d$ as input (line 1). First, the applications are sorted according to ascending request statistics (line 2). Thereafter, we start with an empty

### Algorithm 1 Split applications into pull and push group.

1: **procedure** $\text{PARTITION}(A_d = \{a_1, \ldots, a_n\}, u_d)$
2: \hspace{1em} $A_d \leftarrow \text{sort}(A_d)$
3: \hspace{1em} $C_{\text{UC}}^{\text{pull}} \leftarrow 0$, $C_{\text{MC}}^{\text{push}} \leftarrow u_d \cdot f(n)$
4: \hspace{1em} $k \leftarrow 0$, $C_{\min} \leftarrow C_{\text{UC}}^{\text{pull}} + C_{\text{MC}}^{\text{push}}$
5: \hspace{1em} for $i \leftarrow 1$ to $n$ do
6: \hspace{2em} $C_{\text{UC}}^{\text{pull}} \leftarrow C_{\text{UC}}^{\text{pull}} + 2 \cdot a_i \cdot r_i$, $C_{\text{MC}}^{\text{push}} \leftarrow u_d \cdot f(n - i)$
7: \hspace{2em} if $C_{\text{UC}}^{\text{pull}} + C_{\text{MC}}^{\text{push}} < C_{\min}$ then
8: \hspace{3em} $k \leftarrow i$, $C_{\min} \leftarrow C_{\text{UC}}^{\text{pull}} + C_{\text{MC}}^{\text{push}}$
9: \hspace{1em} end if
10: end for
11: $R_d \leftarrow \{a_1, \ldots, a_k\}$, $U_d \leftarrow \{a_{k+1}, \ldots, a_n\}$
12: **return** $R_d, U_d$
13: **end procedure**
and update rates, respectively. There is no one-to-one correspondence between applications and set the device is a member, respectively. Moreover, there can, thus, improve only half of the pull interaction. And replies still consist of individual unicast messages. This way, data request to all devices of the dynamic set, the device may complicate other aspects such as configuration setup or error handling to ensure a reliable message transport. Nevertheless, we read-

Lemma 1. If $R_d$ and $U_d$ are an optimal pull and push group assignment (according to Eq. 3), then $r_a \leq r_a'$ holds for all applications $a \in R_d$ and $a' \in U_d$.

Proof. Proof by contradiction. Assume $R_d$ and $U_d$ are optimal at costs, but $\exists a \in R_d, a' \in U_d : r_a > r_a'$. Now swap $a$ and $a'$ so that $R_d' := (R_d \setminus \{a\}) \cup \{a'\}$ and $U_d' := (U_d \setminus \{a'\}) \cup \{a\}$. This reduces the pull costs $C_{MC}^{\text{Pull}}(R_d) = C_{MC}^{\text{Pull}}(R_d') + 2r_a - C_{MC}^{\text{Pull}}(R_d \setminus \{a, a'\}) > C_{MC}^{\text{Pull}}(R_d)$. The push costs $C_{MC}^{\text{Push}}(U_d) = u_d \cdot f(|U_d|) = u_d \cdot f(|U_d'|) + 2r_a - C_{MC}^{\text{Push}}(U_d) = 2r_a - C_{MC}^{\text{Push}}(U_d)$ remain constant as the number of multicast receivers has not changed $|U_d'| = |U_d|$. Thus, the initial assignment $R_d$ and $U_d$ was not cost minimal which is a contradiction to the assumption above.

3.4 Heuristics Using Multicast Pull

In Sect. 2.4, we introduced multicast pull as another viable interaction pattern that, however, we have not leveraged in our adaptation strategies yet. This is for a couple of reasons. First, although many middleware implementations enable developers to conveniently communicate with groups of objects or processes, only a few build on efficient network-assisted multicast communication (e.g., IP multicast) as it may complicate other aspects such as configuration setup or error handling to ensure a reliable message transport. Second, the benefit of multicast pull is limited. Despite the cost savings by using multicast to efficiently transmit the data request to all devices of the dynamic set, the device replies still consist of individual unicast messages. This way, we can, thus, improve only half of the pull interaction. And, third, it is not straightforward to trade multicast pull for multicast push as the set of receivers differ, i.e., the devices of a dynamic set and the applications in whose dynamic set the device is a member, respectively. Moreover, there is no one-to-one correspondence between applications and devices while each of them may also have distinct request and update rates, respectively.

To address the last aspect, we need a way to determine the cost fraction of a multicast message attributed to an individual receiver. We estimate this fraction by simply dividing the total costs of the multicast message by the number of receivers. For multicast pull and Eq. 3, we, thus, get

$$C_{MC}^{\text{Pull}}(d) = \frac{r_a \cdot (f(|S_a|) + |S_a|)}{|S_a|} = r_a \cdot \left(1 + \frac{f(|S_a|)}{|S_a|}\right) \tag{11}$$

cost fraction for an individual device $d$ in the dynamic set $S_a$ of application $a$ for $r_a$ pull requests and corresponding replies. Moreover, we can use this estimation as a drop-in replacement for the calculation of the pull costs in the device-based heuristic and the hybrid heuristic as well. We indicate these new variants using multicast pull by a star and call them $h_{dev}^{\text{MC}}$ and $h_{hyb}^{\text{MC}}$, respectively.

For modifying the link-based heuristic that is not using multicast at all, we have to consider the push side too. Likewise, for multicast push and Eq. 3 we get

$$C_{MC}^{\text{Push}}(a) = \frac{u_d \cdot f(|A_d|)}{|A_d|} = u_d \cdot f(|A_d|) \tag{12}$$

cost fraction for $u_d$ multicast data updates pushed to an individual application $a \in A_d$ in whose dynamic set the device $d$ is member. We call the resulting multicast variant of the link-based heuristic $h_{link}^{\text{MC}}$. Please note that for gathering the statistics and calculating the costs correctly, it is now essential to know to how many receivers a multicast message is sent. In our middleware implementation, we, therefore, add this information in the multicast request and data update messages, respectively.

4. EVALUATION

In this section, we evaluate the performance of our adaptive approach. The evaluation is based on discrete event simulations and compares the three heuristics $h_{dev}$, $h_{dev}^{\text{MC}}$, and $h_{hyb}^{\text{MC}}$ with each other and to static communication schemes using either unicast pull or unicast push. $h_{dev}$ switches between unicast pull and unicast push for each link (i.e., for each pair of application and device), $h_{dev}^{\text{MC}}$ switches between complete unicast pull and complete multicast pull for each device, and $h_{hyb}^{\text{MC}}$ uses both partial unicast pull and partial multicast push for each device. Additionally, we also investigate each of the three heuristics in a variant (i.e., $h_{link}^{\text{MC}}$, $h_{link}^{\text{hyb}}$, and $h_{link}^{\text{hyb}}$, respectively) that only uses multicast communication.

Simulations are implemented in Java 1.8. Each simulation run is executed in a single thread in a separate Java VM and spans 150,000 simulation steps in total, where one step represents one millisecond in real time. After a warm-up period of 50,000 ms, the actual measurement time is 100,000 ms. Every 1,000 ms, evaluation routines are executed to trigger strategy switches when and where appropriate. Further, also every 1,000 ms, statistic values are smoothed with a smoothing factor $\alpha$ to lower the impact of older interactions, where $0 < \alpha \leq 1$. For example, $\alpha = 0.9$ means that the new smoothed value is derived by giving the new measured value 90% weight and the old smoothed value 10% weight. All experiments were executed 20 times. Each simulation run is configured with a unique seed such that different setups can be simulated with exactly the same interaction sequence.
The number of applications $|A|$ and devices $|D|$ is set to 500 each for all experiments. Application and devices are randomly placed on nodes with uniform probability. The dynamic set size, i.e., the number of devices bound to an application $a$, is set to $|S_a| = 20$ and the devices bound to a set are randomly chosen with uniform probability. To determine the simulation time at which an application issues the next pull request or a device pushes the next information update, we use exponential distributions with a rate of $\lambda_{set}$ for applications and $\lambda_{act}$ for devices, respectively.

Our heuristics use the estimated communication costs (cf. Sect. 2) to switch from one interaction pattern to another when this promises a cost reduction. However, these costs are only an estimation, as they only take the number of applications and devices as well as getter rates and setter rates into account. The actual communication costs additionally depend on the network topology and on where applications and devices are located in the network. To get a more realistic view on the effectiveness of our heuristic, we use BRITE to generate an Internet-like topology with 1,000 nodes connected by 1,225 links. Based on this topology, we derive the actual resulting communication costs.

For each pair of nodes $(V_{src}, V_{det})$, we determine the shortest paths by a breadth-first-search. For unicast communication, each edge along such a path counts as one message when deriving communication costs. Thus, costs are equal to the length of the path between the source and the destination node. To derive the costs for sending a message to a number of receivers using unicast, the costs are summed up. For multicast communication, we determine a multicast distribution tree spanning the sender and all receivers by joining all shortest paths from the sender to the receivers. The multicast costs are then, equal to the number of links in the tree. This implies that the costs for multicasting a message can be at most as high as the costs to send a unicast message to the same number of receivers. This is the case when the shortest paths to the receivers do not share a single link. Please note that the constructed multicast tree is not optimal. Computing the optimal multicast tree, would require to solve an NP-complete Steinerbaum problem.

### 4.1 Basic Experiments

The basic experiments described in this subsection assign to all applications the same $\lambda_{set}$ rate and to all devices the same $\lambda_{act}$ rate. The range for both rates is $\lambda \in [0.1 \text{s}^{-1}, 10 \text{s}^{-1}]$. As the expected interarrival time is $1/\lambda$ for exponential distributions, a pull of an certain application or a push of a certain device is expected on average every $0.1 \times 100 = 10$ ms and every $10 \times 10,000 = 100$ ms. For the link-based heuristic, we set the threshold $T = 1$, whereas for the other two heuristics, we use $T = 5$.

Figure 3 investigates communication costs for varying request rates $\lambda_{set}$ and update rates $\lambda_{act}$ in a 3-D plot. Communication costs are displayed on the z-axis and include both push messages and pull messages, but also control messages representing the overhead caused by our heuristic. The color of the points indicates the relative share of links in push mode: red indicates that most links are used in pull mode, while blue indicates that most links are used in push mode.

Figure 4a evaluates the link-based heuristic $h_{lnk}$ switching between unicast pull and unicast push independently for each link (i.e., each pair of an application and a device). The plot has a sharp rim indicated by orange points, where unicast pull and unicast push costs are similar and both schemes are applied in the system. By extrapolating the red plane and the blue plain, the cost savings compared to a pure pull and a pure push approach using only unicast communication can be estimated, respectively.

Figure 5a shows the device-based heuristic $h_{dev}$ that switches between complete unicast pull and complete multicast push for all links of a device. The heuristic significantly reduces the number of messages for the majority of points. The reason is that a multicast push message usually saves costs (compared to multiple unicast push messages), because it does not traverse a single link multiple times. The figure also reveals that compared to the link-based heuristic, multicast push is already applied at lower request and higher update rates. This is clearly visible by comparing the rate of the rim in both figures with each other. There is also a small hump in the figure that is most prominent for larger setter rates. This hump is caused by stochastic effects. Without these effects all links should be either in push mode or in pull mode all the time.

Figure 5b evaluates the hybrid heuristics $h_{hyp}$ that uses partial unicast pull and partial multicast push simultaneously, i.e., only a subset of applications communicating with a device receives a multicast push message, while the remaining applications apply unicast pull instead. For most regions of the plot, the communication costs are similar to that of Fig. 5a. However, there is also a region with higher costs indicated by a slightly larger hump. This is again caused by stochastic effects, which have a larger impact in case of the hybrid heuristic. This is because the hybrid heuristic decides for a single link whether it is in push or pull mode, while the device-based heuristic can only use all links either in push or pull mode.

Now, we present the results (cf. Fig. 4) for the variants of the heuristics that solely apply multicast communication. Figure 4b shows (as expected) that the largest change in performance occurs for the link-based heuristic that uses unicast communication for both push and pull before and now uses multicast communication for both schemes. It is obvious that applying multicast communication significantly reduces the communication costs for both the region where push is applied and the region where pull is applied. It can also be seen that pull is applied in a much smaller region than before. This is because push benefits more from multicast communication than pull as in the latter case the reply has to be passed back using unicast in any case. For the other two heuristics (Figs. 4c and 5b), the savings are less prominent, because they already applied multicast push and, thus, only unicast pull was replaced by multicast pull. Two things can mainly been seen in the figures: Firstly, pull is applied in a slightly larger region than before and, secondly, the costs in a large portion of the region where pull is applied are lower than before. However, there is also a small region where the communication costs became higher.

---

1. BRITE: Boston university Representative Internet Topology gEnerator. [https://www.cs.bu.edu/brite/](https://www.cs.bu.edu/brite/)
Looking again at Figs. 4c and 4e, the results of the two variants of the hybrid heuristic \( h_{hyb} \) and \( h_{hyb}^\ast \) seem disappointing at a first glance. However, we used the same getter rate for all applications and the same setter rate for all devices, which is the worst-case for the hybrid heuristic. Indeed, for the hybrid heuristic to perform better than the device-based heuristic, it is necessary that the getter rates of the applications and/or the setter rates of the devices vary. Only in this case, the higher flexibility of the hybrid heuristic is expected to pay out. Therefore, we consider spread getter rates in the next subsection. For spread setter rates, similar results can be obtained.

### 4.2 Spread Getter Rates

So far, we assumed that all applications have equal getter rates. This made choosing the right strategy difficult, especially for the hybrid heuristic. Therefore, we now investigate the effect of spread getter rates. To model spreading, we consider a minimal getter rate \( \lambda_{get}^{min} \), a maximal getter rate \( \lambda_{get}^{max} \), and a spreading coefficient \( i \), where \( 0 \leq i \leq 10 \).

The basic idea is that for \( i = 0 \), all applications get the same getter rate. For \( i \geq 1 \), each application gets one of two getter rates assigned, either a lower getter rate \( \lambda_{get}^l \) or a higher getter rate \( \lambda_{get}^h \). We use \( \lambda_{get}^l = E(\lambda_{get}) - i \cdot \Delta \) and \( \lambda_{get}^h = E(\lambda_{get}) + i \cdot \Delta \) so that the difference among the two getter rates grows with increasing \( i \) and that for \( i = 10 \), both rates equal \( \lambda_{get}^{min} \) and \( \lambda_{get}^{max} \), respectively.

If \( E(\lambda_{get}) \) lies in the middle of the interval \([\lambda_{get}^{min}, \lambda_{get}^{max}]\), \( \Delta = (\lambda_{get}^{max} - \lambda_{get}^{min})/20 \) is used and the two getter rates are chosen with equal probability of 0.5. This ensures that the expected getter rate \( E(\lambda_{get}) \) computed over all applications stays at the same value when \( i \) is varied from 0 to 10. If, however, \( E(\lambda_{get}) \) is not in the middle of the interval \([\lambda_{get}^{min}, \lambda_{get}^{max}]\), two different delta values are used. Then, \( \lambda_{get}^l = E(\lambda_{get}) - i \cdot \Delta_l \) and \( \lambda_{get}^u = E(\lambda_{get}) + i \cdot \Delta_u \), where \( \Delta_l = (E(\lambda_{get}) - \lambda_{get}^{min})/10 \) and \( \Delta_u = (\lambda_{get}^{max} - E(\lambda_{get}))/10 \). Furthermore, both getter rate values must be sampled with different probabilities to ensure that \( E(\lambda_{get}) \) stays at the same value. The probability \( p_l \) to choose the lower getter rate \( \lambda_{get}^l \) is \( p_l = \frac{\Delta_l}{\Delta_l + \Delta_u} \), while the probability \( p_u \) to choose the upper getter rate \( \lambda_{get}^u \) is \( p_u = \frac{\Delta_u}{\Delta_l + \Delta_u} = 1 - p_l \).

#### 4.2.1 Link-based Heuristic

Figure 5a shows the results of spread getter rates for the link-based heuristic \( h_{link} \). It can be seen that heuristic \( h_{link} \) saves the more communication costs compared to pure unicast push and pure unicast pull, the higher the spread among the two getter rates becomes. In contrast to that and as expected, the costs of pure unicast push and pure unicast pull do not change when the spreading coefficient \( i \) varies as \( \lambda_{set} \) and \( E(\lambda_{get}) \) do not change. Finally, for smaller values of \( i \) and \( \lambda_{set,2} \) or \( \lambda_{set,3} \), the costs of \( h_{link} \) is a little bit higher than the costs of pure unicast push. This is because some links were switched to unicast pull due to stochastic effects although unicast push would raise lower costs.

Figure 6a shows the results of evaluation of spread getter rates for the link-based heuristic \( h_{link} \) that uses multicast.
push and multicast pull instead of unicast communication. By using multicast communication, the communication costs are reduced to about a third of the costs of \( h_{\text{tink}} \) that applied unicast communication. Furthermore, heuristic \( h^*_{\text{tink}} \) saves the more communication costs compared to pure multicast push and pure multicast pull, the higher the spread among the two getter rates becomes. And again, costs of pure multicast push and multicast pull do not change when the spreading coefficient is varied.

### 4.2.2 Device-Based and Hybrid Heuristic

Figure 5a shows the results of the evaluation of the spread getter rates for the device-based heuristic \( h_{\text{dev}} \), and the hybrid heuristic \( h^*_{\text{hyb}} \), while Fig. 5b shows the results for their pure multicast variants \( h^*_{\text{dev}} \), and \( h^*_{\text{hyb}} \), respectively. In both figures, the hybrid heuristic performs the better (when compared to the device-based heuristic) the higher the spreading coefficient \( i \) becomes. This shows that the hybrid heuristics \( h^*_{\text{hyb}} \) and \( h^*_{\text{hyb}} \) are able to exploit spread getter rates. In contrast to that, the communication costs of \( h_{\text{dev}} \) and \( h^*_{\text{dev}} \) only slightly decrease for increasing \( i \). However, this effect is due to stochastic variations. When looking at both figures, it can be seen that the communication costs of \( h^*_{\text{hyb}} \) is only a little bit lower than those of \( h^*_{\text{dev}} \). Thus, enabling multicast pull does not change the costs to a larger extend in the investigated scenario. The same is true for \( h^*_{\text{dev}} \) and \( h^*_{\text{dev}} \).

### 4.3 Multicast Cost Functions

The heuristics presented rely on a cost function for estimating the communication costs of multicast and unicast communication. The estimation is solely based on the number of receivers and incorporates only a rough knowledge about the network topology and where applications and devices are located. Thus, we suspect that the estimation may be better or worse depending on the actual topology and the applied cost function. We, therefore, investigate the effect of different multicast cost functions on the effectiveness of the device-based and the hybrid heuristics, which are applying multicast push. As cost functions, we decided to use \( f_1(n) = n^{0.75} \), \( f_2(n) = n^{0.5} \), and \( f_3(n) = n^{0.25} \), where \( n \) is the number of receivers.

For investigating the effects of different multicast cost functions, we present results for two slices (one that varies the
getter rate and one that varies the setter rate). We chose the two slices such that they allow a detailed investigation of the area, where the switch from push to pull and vice versa should occur. For f2(n), the plot of the slice is derived from Figs. 8 and 7. The other plots show the results for the other two cost functions which were newly computed. The results are shown in Figs. 7 and 8, which are discussed next. For a better comparability, the x-axis in the plots that investigate varying getter rates (i.e., Figs. 7a and 8a) have been mirrored such that the 0 is on the right side of the x-axis.

4.3.1 Original Heuristics

Figures 7a and 8a depict the overall communication costs for the original heuristics hdev, and hhyb for the three considered multicast cost functions. In Fig. 7a, the getter rate is varied and a fixed setter rate of λset = 0.01 is used. What can be seen in the figure is that the choice of the multicast cost function indeed has an impact on the effectiveness of our adaptive approach. Interestingly, the difference between the device-based and the hybrid heuristics are rather small. When the getter rate is below 0.002, the cost function f1 = n^0.75 leads to the best results, but it delivers the worst result for larger getter rates. The reason is that it switches way too late from pull to push causing high costs when the getter rate is between 0.002 and 0.007. The other two cost functions f2 = n^0.5 and f3 = n^0.25, switch early enough to push, but raise higher costs than f1 for lower getter rates. This is because some applications switch to push too early. Considering both areas, f2 seems to be the best choice.

In Fig. 8a, the setter rate is varied and a fixed getter rate of λget = 0.001 is used. It can be seen that f3 = n^0.75 fails to switch from push to pull when the setter rate is increased, while f2 = n^0.5 switches to pull although too late. Cost function f1 = n^0.75 seems to be the best choice here, followed by cost function f2. Considering both figures together, f2 seems to be the best choice.

4.3.2 Heuristics solely using Multicast

Figures 7b and 8b depict the overall communication costs for heuristics h^∗dev and h^∗hyb which solely apply multicast communication for the three considered multicast cost functions. In Fig. 8a, the getter rate is varied and a fixed setter rate of

\[ \lambda_{get} = 0.01 \]

leads to the best results, but it delivers the worst result for larger getter rates. The reason is that it switches way too late from pull to push causing high costs when the getter rate is between 0.002 and 0.007. The other two cost functions f2 = n^0.5 and f3 = n^0.25, switch early enough to push, but raise higher costs than f1 for lower getter rates. This is because some applications switch to push too early. Considering both areas, f2 seems to be the best choice.

In Fig. 8b, the setter rate is varied and a fixed getter rate of λset = 0.001 is used. It can be seen that f3 = n^0.75 fails to switch from push to pull when the setter rate is increased, while f2 = n^0.5 switches to pull although too late. Cost function f1 = n^0.75 seems to be the best choice here, followed by cost function f2. Considering both figures together, f2 seems to be the best choice.
For the fitted multicast cost functions, we evaluate $g_1 = a_1 \cdot n^{0.75} + b_1$ and $g_2 = a_2 \cdot n^{0.5} + b_2$, while for the unicast cost function, we evaluate the linear cost function $h(n) = a_3 \cdot n + b_3$. Then, we compare the fitted multicast cost functions together with the fitted unicast cost function to the original ones where $a_1 = a_2 = a_3 = 1$ and $b_1 = b_2 = b_3 = 0$.

The results for the original hybrid heuristic $h_{hyb}$ are plotted in Figs. 9a and 9b. In Fig. 9a, where the getter rate $\lambda_{get}$ is varied, it can be seen that the fitted cost function $g_1 \propto n^{0.75}$ performs better than the original cost function $f_1$ in the push area. However, it performs slightly worse in the pull area. It can also be seen that the fitted cost function $g_2 \propto n^{0.5}$ switches later from push to pull performing worse than the original cost function $f_2$. However, it performs slightly better in the pull area. In Fig. 9b where the setter rate $\lambda_{set}$ is varied, it can be seen that the fitted cost function $g_1 \propto n^{0.75}$ also performs better than the original cost function $f_1$ in the push area, but worse in the pull area. The fitted cost function $g_2 \propto n^{0.5}$ performs better in the pull area than the original cost function $f_2$, but worse in the push area.

The results for the new hybrid heuristic $h_{hyb}^*$ solely using multicast are plotted in Figs. 10a and 10b. In Fig. 10a, where the getter rate $\lambda_{get}$ is varied, it can be seen that, the
fitted cost function \( g_1 \propto n^{0.75} \) performs better in the push area than the original cost function \( f_1 \), but worse in the pull area. The fitted cost function \( g_2 \propto n^{0.5} \) performs better in pull area, but worse in the push area than the original cost function \( f_2 \). In Fig. 10b, where the setter rate \( \lambda_{\text{set}} \) is varied, it can be seen that the fitted cost function \( g_1 \propto n^{0.75} \) performs better than the original cost function \( f_1 \) in the push area, but worse in the pull area. Moreover, the fitted cost function \( g_2 \propto n^{0.5} \) performs better in the pull area, but worse in the push area than the original cost function \( f_2 \).

Summarizing, using linearly fitted cost functions has no clear advantage over using the original cost functions. Usually when a cost function has an advantage in the push area, it performs worse in the pull area and vice versa. Maybe a different fitting model could lead to better results.

5. RELATED WORK

In the Internet of Things (IoT) domain, several prominent protocols are competing for the favor of the developers. For example, MQTT [2], CoAP [13], and XMPP [12], just to name a few. Despite the differences and details, they all address the developers’ need for flexible interaction by offering versatile group communication capabilities. Either these are key features inherent to the protocol (e.g., publish/subscribe in MQTT) or have been added by extensions. In case of CoAP, group communication and multicast support is motivated by the example of a single message being able to immediately switch on all lights in a given room [11]. In fact, this perfectly suits our dynamic set concept. However, none of the protocols is adaptive by design, i.e., able to automatically switch communication paradigms at runtime when beneficial. Instead, it is left to the developer to anticipate the number of applications and devices as well as their request and update rates and to choose the best communication scheme (i.e., pull or push) at design time.

Several adaptive communication schemes have been presented in literature, in particular, for the dissemination of Web data. Franklin and Zdonik [1] explore the design space for information dissemination (e.g., client push/server pull, unicast/multicast, event-driven aperiodic/scheduled periodic). Deolase et al. [3] present adaptive push/pull schemes for clients and intermediary proxies that can switch between both modes or use them simultaneously. This ensures a certain degree of data consistency and improves the application resiliency while limiting arising network costs.

Likewise, for caching and cloud environments, the authors of [16] and [5], respectively, relax data consistency constraints in order defer and safe data requests and/or update notifications which leads to cost reductions. Minson and Theodoropoulos [7] apply machine learning techniques to better predict update rates and, thus, tune the delivery scheme. However, obtained results are not significantly better when compared to much simpler adaptation schemes. Although relaxing consistency constraints and predicting update rates are orthogonal to our approach, their addition/integration is not straightforward as none of them has been designed for group communication and multicast delivery in the first place. Acharya et al. [4] show how to balance request and update rates when using a high-bandwidth frontchannel for broadcasting data and a small-bandwidth backchannel for pull requests.

Though very interesting, the results are tightly bound to the asymmetric setting as well as the use of a broadcast media.

6. CONCLUSIONS

A dynamic set bundles a set of objects and allows an application to transparently access these devices through a proxy having the same interface as a single of these objects. This approach allows an application to deal flexibly with a varying number of devices available at runtime bridging the gap between design time and runtime.

In this paper, we presented adaptive approaches that aim at reducing the communication costs caused by polling in the context of dynamic sets. We proposed three basic heuristics and their respective pure multicast variants that switch between different communication schemes at runtime. The link-based heuristics \( h_{\text{dev}} \) and \( h_{\text{hyb}} \) that compare push and pull rates for each pair of object and application convince by their simple design and their excellent results in a unicast environment and a solely multicast setting, respectively. If unicast and multicast communication are mixed, it is beneficial to disseminate frequent data updates via multicast messages using the \( h_{\text{dev}} \) or the \( h_{\text{hyb}} \) heuristic. From these two heuristics, \( h_{\text{hyb}} \) is more complex but achieves better results as it can apply multicast push to deliver information updates to those applications having a sufficiently large getter rate, while the other applications with lower getter rates can still apply unicast pull. The pure multicast variants \( h_{\text{dev}}^* \) and \( h_{\text{hyb}}^* \), however, do not perform better than \( h_{\text{dev}} \).

All heuristics and their variants adapt the communication pattern based on gathered statistics, i.e., without global knowledge of the exact network topology. The evaluation showed that a simple cost model roughly approximating the network topology is sufficient to make reasonable decisions for adaptations. After fitting the model to the actual topology, we were able to fine-tune the heuristics, but could not gain substantial further savings, i.e., cost reductions comparable to those achieved by the basic heuristics and their variants with respect to a static communication scheme. In fact, these results show that, despite detailed knowledge of the network layer, adaptation schemes implemented on application level can be beneficially applied in a broad range of dynamic environments.

For future work, we want to analyze our heuristics for varying getter and setter rates that quickly change over time. In particular, we want to explore advanced smoothing schemes and thresholds that increase the heuristics responsivity, but do not tend towards oscillations. Furthermore, we are planning to integrate the heuristics into a publish/subscribe middleware. This allows to place intermediate nodes in the topology that may also switch between push and pull enabling even more flexible communication patterns that presumably reduce communication costs further.

7. REFERENCES


Matthias Prellwitz is a PhD student at the Architecture of Application Systems group at University of Rostock since 2013. He studied computer science at University of Applied Sciences (HTW) in Berlin, Germany, and at Old Dominion University in Norfolk VA, USA. In 2010, he received his master's degree. His research focuses on programming abstractions as well as adaptivity aspects of distributed systems.

Helge Parzyjegla is a postdoctoral researcher at the Architecture of Application Systems group at the University of Rostock. He studied computer science at the Berlin University of Technology. In 2013, he received a PhD in computer science from the University of Rostock. His research interests lie in the area of distributed, event-driven systems, publish/subscribe middleware, and network softwarization with the focus on aspects of self-organization and adaptive behavior.

Gero Mühl has studied computer science and electrical engineering at University of Hagen. In 2002, he received a PhD in Computer Science from Darmstadt University of Technology. From 2002 to 2009 he was a research assistant at the Communication- and Operating Systems group at Berlin University of Technology, where he was awarded with the habilitation in 2007. Since 2009 he is head of Architecture of Application Systems group at University of Rostock. His research interests lie in the fields of self-organizing distributed systems, event-driven architectures, and adaptive storage systems.
Functional Feasibility Analysis of Variability-Intensive Data Flow-Oriented Applications over Highly-Configurable Platforms

Sami Lazreg  
Visteon Electronics and  
Université Côte d’Azur, CNRS,  
I3S, France  
slazreg@visteon.com

Philippe Collet  
Université Côte d’Azur, CNRS,  
I3S, France  
collet@i3s.unice.fr

Sébastien Mosser  
Université Côte d’Azur, CNRS,  
I3S, France  
mosser@i3s.unice.fr

ABSTRACT
Data-flow oriented embedded systems, such as automotive systems used to render HMI (e.g., instrument clusters, infotainments), are increasingly built from highly variable specifications while targeting different constrained hardware platforms configurable in a fine-grained way. These variabilities at two different levels lead to a huge number of possible embedded system solutions, which functional feasibility is extremely complex and tedious to predetermine. In this paper, we propose a tooled approach that capture high level specifications as variable dataflows, and targeted platforms as variable component models. Dataflows can then be mapped onto platforms to express a specification of such variability-intensive systems. The proposed solution transforms this specification into structural and behavioral variability models and reuses automated reasoning techniques to explore and assess the functional feasibility of all variants in a single run. We also report on the validation of the proposed approach. A qualitative evaluation has been conducted on an industrial case study of automotive instrument cluster, while a quantitative one is reported on large generated datasets.

CCS Concepts
• General and reference → Design; Validation;  
• Computer systems organization → Embedded systems;  
• Software and its engineering → Software product lines;  
• Theory of computation → Verification by model checking;

Keywords
Embedded system design engineering; variability modeling; feature model; behavioral product lines model checking.

1. INTRODUCTION
Validating embedded systems design at early stages of development is of fundamental importance in industry. Ideally embedded system design should be modeled from high-level specifications, and then assess against possible implementations. Data-flow oriented embedded systems, such as automotive systems used to render HMI (e.g., instrument clusters, infotainments) are typically built from highly variable specifications. They are composed of a data-flow driving and feeding graphical processors to provide efficient and high-quality graphic rendering at a lower cost, while targeted hardware platforms are composed of heterogeneous and constrained hardware components. The variability is then two-fold, with multiple graphic data-flow variants that can meet functional requirements, and diverse targeted hardware platform, which are highly configurable in a fine-grained way. These dimensions of variability dreadfully increase the size of the design space of these embedded systems (i.e., the number of possible embedded system implementation designs), making the feasibility assessment of these systems extremely tedious and complex.

Facing this issue, we determine three challenges to be tackled: (i) capturing and modeling from high-level specifications, structure, behavior and variability of these embedded systems (e.g., data-flow and platform alternatives, data sizes, memory capacities, graphic pipelines), (ii) inferring automatically all possible embedded system design implementations from specification models and, (iii) exploring and assessing the feasibility of all system implementations w.r.t. the predefined structural, behavioral and variability constraints. Current approaches [22, 15, 16] assess functional feasibility of constrained data-flow-oriented embedded systems, but do not capture nor manage variability at both levels. Some Ad hoc techniques are trying to handle either platform variability (as reconfigurable architectures [21, 20] [18]) or functional variability (as multiple scenarios [19, 25] or multi-modes systems [26, 17]). On the other hand,
approaches tackling both kinds of variability [12] are focusing on optimal platform selections to implement multiple functional variants at a lower cost, but they do not manage structural and behavioral properties (e.g. data sizes, memory capacities, graphic pipelines).

In this paper we propose an approach that extends these researches by supporting a complete modeling and assessment of structural, behavioral and variability properties of the targeted embedded systems by combining embedded system design engineering [22, 15, 16] and Product line engineering techniques [1, 8]. The proposed solution is model driven and i) captures high level variable data-flow and platform specifications following principles of separation of concern, ii) maps variable data-flow requirements into a description of the targeted variable hardware platform, so to infer the embedded system design space (i.e. all system implementations), iii) transforms the design space into a behavioral product line to reuse automated reasoning techniques (i.e. SAT solving, variability-aware model checking) to explore and assess the functional feasibility of all system design implementations in a single run. The framework also allows to remove invalid designs from the design space by constraining it.

This paper is an extended version from a paper published in the Variability and Software Product Line Engineering track of the SAC 2018 conference. In this extension, we detail a complete evaluation of the proposed approach with:

- complete implementation details with end-to-end sample materials,
- a qualitative evaluation on a real industrial use case in automotive systems, i.e., an instrument cluster product line,
- a quantitative evaluation on the scalability of the approach, using large generated datasets on both application and platform sides,
- a discussion on the threats to validity.

The remainder of the paper is organized as follows. Section 2 introduces the context and motivations illustrated by a running example. Section 3 presents the proposed framework, detailing each model and step. Section 4 details the qualitative and quantitative validation, and discusses threats to validity, while Section 5 concludes the paper.

2. MOTIVATIONS

Requirement gathering and validation of this research work have been realized in the context of an industrial collaboration with Visteon Electronics, a world class leader in automotive systems (e.g. instruments clusters, infotainment, connected vehicles). In the following we introduce one of the company case studies, extract a running example, determine requirements from them and discuss related work.

2.1 Case study

The case study is focusing on functional validation of some instrument clusters. By applying various data-flow image processing effects, such as blending, warping and scaling, an instrument cluster system improves driver experience with useful and high quality 2D/3D Human-Machine Interface (HMI). The embedded hardware platforms used to develop these systems are more and more highly configurable, but constrained in terms of architecture and capacities. Furthermore, multiple graphic data-flows variants can meet the client HMI requirements, but they also depend on the platform architectures and capacities. We consider this case study as representative of variability-intensive data-flow oriented systems. Different forms of variability, from high-level data-flows to low-level platforms lead to a huge number of possible system solutions, which feasibility is extremely complex and tedious to predetermine early in the development process.

2.2 Running Example

We now introduce a running example of a simplified instrument cluster. The data-flow on Fig. 1 represents different image flow processing that meet the HMI functional requirements. Images are processed by graphical tasks: image $d_i$ has two different possible resolutions (e.g. 800x480, 480x320) and will be processed by task $C$. Image $d_i$ can be either processed by task $A$ or task $B$.

![Figure 1: Functional specification](image)

On the hardware side (cf. Fig. 2), the platform provides image processing capabilities through non programmable pipeline processors of DCU (Display Controller Unit) or GPU (Graphic Processing Unit) type, as well as data storage functionalities through RAM (Random Access Memory) and ROM (Read Only Memory). RAM and GPU are optional in the actual hardware products, so a system implementation may contain or not these components. Among variabilities in platforms, one can write data into and/or read data from RAM memory while only reading data is possible from ROM. Moreover, memory storage have limited and possibly variable (e.g. RAM) capacity. Contrary to a DCU, which renders directly processed images into display, a GPU needs to store its processing result into RAM. Graphical hardware processors are often designed as a multi-step pipeline, composed of several hardware implemented processing steps, and processor internal fifo memory buffers transferring data from one step to another. In our example, while a GPU can apply $A$ followed by $B$ processing on images in a single pass, a DCU can apply $A$, then followed by $C$.

Images and processing could be, respectively, stored and pro-
and assess the resulting design space. Assessing the different automatically map data-flows on platforms in order to infer platform specifications, these approaches are not capable of behavioral aspects. However, given high-level data-flows and variability-aware model-checking, temporal properties and safety-critical systems or protocols, and to validate, through correctly processor pipelines and memories, and (iii) variability constraints such as component dependency and exclusion. In our example, the application data-flow has 4 variants while the Platform exposes 3 architecture variants. Even with this simplified case, this leads to 178 possible implementations, in which 58 satisfy constraints and could be developed by engineers.1

2.3 Related Work

In the context of our work, engineers must be assisted to assess the functional feasibility of the different potential embedded system solutions, with means to capture both the high-level functional requirements and the specifications of targeted variable platforms. Ideally, a solution should be able to capture structural, behavioral and variability properties of both functional and platform specifications at a fine-grained level, so to use these input models to automatically infer all possible embedded system implementation designs and assess the resulting consistent design space.

In the product line engineering, lots of approaches [1, 8] are capable to model variable transition-based systems such as safety-critical systems or protocols, and to validate, through variability-aware model-checking, temporal properties and behavioral aspects. However, given high-level data-flows and platform specifications, these approaches are not capable of automatically map data-flows on platforms in order to infer and assess the resulting design space. Assessing the different mapping manually is not feasible in practice, as the activity would be tedious and error-prone.

In embedded system design engineering, most of the approaches capture high-level application and platform specifications, and map an application on hardware platforms in order to find, by design space exploration, an optimized system implementation for a single functional specification on a single platform [22, 15]. Consequently, they do not capture nor manage variability at the application level and hardware platform variability is limited to component capacities (e.g. memory and bus size, processor frequency). Some other approaches try to handle some limited variability in functional specifications (e.g. optional task, alternative tasks, variable data) (as multiple scenarios [19, 25] or multi-modes systems [26, 17]), but they do not manage platform variability. Some others try to handle some limited variability in platforms (e.g. optional resource, resource dependency, memories sizes) with reconfigurable architectures [21, 20] [18]. To the best of our knowledge, none of these approaches handle variability in both application and platform sides so to assess the feasibility of our class of problem.

Interestingly, the recent approach of Graf et. al. [13, 14] manages some variability in both platform and functional specifications. On the platform variability side, resource components can be selected or not, while optional and mutually exclusive task groups are managed on the functional part. However, the approach is handling a coarse-grain form of requirements and cannot capture some of our specifications (e.g.data and memory sizes, as well as platform aspects such as processor pipelines or fifo buffers). Additionally, only structural validation of the system implementations is supported. Behavioral properties (e.g. data sizes, memory capacities, graphic pipelines) and behavioral constraints (e.g., absence of deadlock, reachability, liveness, safety etc.), which are fundamental in our case, cannot be checked.

3. PROPOSED FRAMEWORK

3.1 Overview

The proposed approach follows a model driven decomposition, based on the well-known robust Y-Chart pattern [2, 16], which separates application and platform into different concerns. This allows modular specification and reasoning about the different parts of the specified embedded systems. Given high-level variable dataflow and platform inputs that notably captures the variability of functional and platform specifications, the framework will i) map all implementation of each data-flow variants on each platform configuration, ii) generate a Featured Transition System (FTS) [8] from the system design space model, i.e. representing system implementations (cf. Fig. 3). This model consists in an extended form of automaton product line, which is then iii) checked in one run by a variability-aware model checker.

As shown on Fig. 3, our framework consists of three main models and two processes. We give here an overview while the following sections will detail and illustrate these elements.

Variable applications: a functional expert is in charge of capturing the functional requirements (cf. fig. 1) of the em-

![Figure 2: Platform specification](image)
bedded system through an extended data-flow (cf. sec. 3.2). This model contains the classic structure and behavior of the data-flow (data, task, data-path, etc.), but also captures the variability in both structural properties (e.g., data size) and behavioral properties (e.g., alternative flows).

**Variable platforms**: on this side, a platform expert is in charge of expressing the platform specification (cf. fig. 2) as a templated component based system (cf. sec. 3.3). This model contains a set of components connected with each others. Similarly to the application one, the platform model also captures the variability of the defined components.

**Variability-aware mapping process**: the mapping algorithm (cf. sec. 3.4) consumes the application and platform input models previously defined and generates the Variability-Intensive Design Space, i.e. representing system implementations. It is made of two steps: (i) it finds for each task data and data-path (cf. fig. 1) all the possible mappings on appropriate platform processors and storage (cf. fig. 2); basically this is done by matching the task names with the names of the hardware functions of processors; data-paths are mapped on reachable memory of appropriate processor hardware functions; (ii) as the design space contains all mapping possibilities, the algorithm prunes unfeasible mappings w.r.t. structural and variability constraints.

**Design space as a behavioral product line**: from the system design space model, a Behavioral Product Line representing all system implementations is generated (cf. sec. 3.5). This product line is represented as a featured automaton, so that we can reuse and adapt techniques that rely on variability-aware model-checking to validate the inferred systems. The basic idea is to transform a variable data-flow, a variable platform, and mappings to a data-flow automaton using, through a mapping automaton, a platform automaton to execute it. Valid executions of the application automaton should then respect generated properties representing end state reachability to ensure that the execution is correctly scheduled and executed onto the platform automaton.

**Validation process**: the validation process reuses automated reasoning techniques to assess structural and behavioral functional feasibility of the system variants represented by the behavioral product line (cf. sec. 3.6). The model checking is going to determine classic properties, such as safety, absence of deadlock, and our state reachability generated property, on all variants in one run. As a result, the validation solves and extracts valid variants respecting all structural, behavioral and variability constraints.

### 3.2 Applications as Variable Data-Flows

In our approach, a functional expert captures structure, behavior (data, task, data-path, etc.) and variability aspects (data size, alternative flows, etc.) of the functional requirements of the embedded system through an extended data-flow model. The extensions concern variability of data and data aspects of functional requirements, and in the following, we propose a formal data-flow model to do so.

**Definition 1.** A variable data-flow graph is a tuple $VDG = (T, D, Path, E, \zeta)$ where:

- $T$ is a set of tasks,
- $D$ is a set of source data, and, $\zeta : D \rightarrow \{s_0, ..., s_i \in \mathbb{N}^*\}$ returns a set of alternative sizes of data,
- $|\zeta(d)\in D| = \begin{cases} >1, & \text{if } d \text{ has a variable size} \\ 1, & \text{if } d \text{ has not a variable size} \end{cases}$
- Path is a set of data-paths by which producer and consumer (i.e. tasks and data) are connected.
- $E \subseteq (T \cup D) \times Path \times T$ is the set of edges representing flows processing between producers and consumers.

The set of connected, input data-paths to a task $I(t)$, output data-paths from a task or data $O(v)$ are denoted by:

$\{p \in Path \mid (x, p, t) \in E\}$,
To capture variability aspects of a platform specification, we constraints on resources (dependency, incompatibility, etc.).

A variable data-flow represents multiple data-flow variants. To implicitly represent all these variants in a single model, we follow the same approach as in variable workflows from [13], allowing data-paths to have multiple input and output tasks connected.

A data-path can be connected to multiple alternative, input tasks if \(|I(p)| > 1\), and output tasks if \(|O(p)| > 1\). But, if \(|I(p)| = 1 \land |O(p)| = 1\), the data-path is connected to only one input and output task (i.e. the data-path has no flow variability).

As data can have alternative sizes, we introduce the function \(\zeta\) which returns the set of alternative sizes \(S = \zeta(d)\), each datum has at least one size and if \(|\zeta(d)| > 1\) the size of \(d\) is variable.

If the data-flow has no flow variability, \(\forall p \in \text{Path}, |I(p)| + |O(p)| = 2\,\text{ and no data variability, } \forall d \in \text{D}, |\zeta(d)| = 1\,\text{ the data-flow is not variable.}

The functional specification of the running example \(VDG_{\text{re}}\) is then represented as
\[
(T_{\text{re}} = \{a,b,c\}, D_{\text{re}} = \{d_1,d_2\}, \text{Path}_{\text{re}} = \{p_1,p_2,p_3\}, \\
E_{\text{re}} = \{(d_1,p_1,a),(a,p_2,c),(d_1,p_1,b),(b,p_2,c),(d_2,p_3,c)\})
\]

with,
\[
\zeta(d_1) = 512, \zeta(d_2) = 512, 1024, \\
I(p_1) = d_1, I(p_2) = \{a,b\}, I(p_3) = d_2, \\
O(p_1) = \{a,b\}, O(p_2) = c, O(p_3) = c, \\
I(a) = I(b) = p_1, I(c) = \{p_2,p_3\}, \\
O(a) = O(b) = p_2, O(c) = \emptyset, O(d_1) = p_1, O(d_2) = p_3.
\]

### 3.3 Platforms as Variable Resource Graphs

A variable platform specification is represented by a templated component based system (multi-pass processors, streaming processor, read-only memory, read-write memory, write-only memory, first-in-first-out buffers etc) where platform can have optional resource components and variability constraints on resources (dependency, incompatibility, etc.). To capture variability aspects of a platform specification, we propose a formal architecture model defined as follows.

**Definition 2.** A variable resource graph is a tuple \(VRG = (Proc, S, C_0, \xi, \theta, \phi_{\text{requires}}, \phi_{\text{excludes}})\) where:

- \(Proc = (F, B, C_0 \subseteq (F \times B) \times (B \times F))\) is a processor composed of a set \(F\) of possible functions, \(B\) is a set of processor internal first-in-first-out buffers and \(C_0\) the connections between the different functions and buffers representing the processor pipeline.

- \(S\) is a set of memory storage, and \(\xi : S \rightarrow \{c_0,...,c_i \in \mathbb{N}\}\) returns a set of alternative capacities of storage \(s \in S\),

\[
|\xi(s)| = \begin{cases} 
> 1, \text{if } s \text{ has a variable storage capacity} \\
1, \text{if } s \text{ has not a variable storage capacity}
\end{cases}
\]

- \(C_0 \subseteq (S \times \text{Proc}) \cup (\text{Proc} \times S)\) is the set of connections between memory storage and processors,

- \(R \subseteq \text{Proc} \cup S\) is the set of resource components (i.e. processors and memory storage),

- \(\theta : R \rightarrow \mathbb{B}\) return true if a component (i.e. processor or memory storage) is optional,

- \(\phi_{\text{requires}} : R \rightarrow R\) captures dependency between resource components, similarly \(\phi_{\text{excludes}} : R \rightarrow R\) captures incompatibility.

The set of input memories to a processor function \(I(f)\), output memories from a processor function \(O(f)\) are denoted by:

\[
\exists p = (F,B,C_0) \in \text{Proc}, \\
I(f \in F) : \{m \in S \cup B|(m,p) \in C_0 \lor (m,f) \in C_0\}, \\
O(f \in F) : \{m \in S \cup B|(p,m) \in C_0 \lor (f,m) \in C_0\}.
\]

As a variable platform represents multiple platform configurations, we also capture implicitly all these configurations by introducing several functions, \(\theta\) manages the optionality of a resource component. If \(\theta(r) = \bot\) the resource is mandatory, otherwise the resource is optional, \(\phi_{\text{requires}}\) and \(\phi_{\text{excludes}}\) manages constrained relations of dependency and incompatibility between resource components. \(\phi_{\text{requires}}(r) = r_0\) means that if \(r\) is implemented then \(r_0\) must be implemented too. \(r\) depends on \(r_0\). On the contrary \(\phi_{\text{excludes}}(r) = r_0\) means that \(r\) and \(r_0\) cannot be implemented on the same platform variant. \(r\) and \(r_0\) are alternatives.

As memory storage can have alternative capacities, we introduce the function \(\xi\) which returns the set of alternative capacities \(C = \xi(s)\), each memory storage has at least one size and if \(|\xi(s)| > 1\) the capacity of \(s\) is variable.

If the platform has no component variability \(\forall r \in R, \theta(r) = \bot\) and no variable memory storage, \(\forall s \in S, |\xi(s)| = 1\) the platform is not variable.

The platform specification of the running example \(VG_{\text{re}}\) is then formalized as
\[
(Proc_{\text{re}} = \{\text{DCU, GPU}\}, S_{\text{re}} = \{\text{RAM, ROM}\}, \\
C_{\text{re}} = \{(\text{RAM, DCU}), (\text{ROM, DCU}), (\text{RAM, GPU}), (\text{ROM, GPU}), (\text{GPU, RAM})\})
\]

where,
\[
\text{DCU} = \{a_{\text{dcu}}, c_{\text{dcu}}\}, B_{\text{dcu}} = r_0_{\text{dcu}}, \\
C_{\text{dcu}} = \{a_{\text{dcu}}, r_0_{\text{dcu}}, (r_0_{\text{dcu}}, c_{\text{dcu}})\}, \\
\text{GPU} = \{a_{\text{gpu}}, b_{\text{gpu}}\}, B_{\text{gpu}} = r_0_{\text{gpu}}, \\
C_{\text{gpu}} = \{a_{\text{gpu}}, r_0_{\text{gpu}}, (r_0_{\text{gpu}}, b_{\text{gpu}})\}.
\]
with,
\[ \xi(\text{ROM}) = 4096, \xi(\text{RAM}) = \{1024, 2048\}, \]
\[ \theta(\text{GPU}) = \theta(\text{RAM}) = \top, \theta(\text{DCU}) = \theta(\text{ROM}) = \bot \]
\[ \phi_{\text{requires}}(\text{GPU}) = \text{RAM}, \phi_{\text{requires}}(\text{RAM}) = \emptyset, \]
\[ I(\text{a}_{\text{gpu}}) = \{\text{ROM}, \text{RAM}\}, O(\text{a}_{\text{gpu}}) = \{\text{r}_{\text{0}}, \text{RAM}\}, \]
\[ I(\text{e}_{\text{dcu}}) = \{\text{r}_{\text{0}}, \text{RAM}, \text{ROM}\}, O(\text{e}_{\text{dcu}}) = \emptyset. \]

3.4 Variability-Aware Mapping Process

The mapping algorithm takes as inputs the variable data-flow and configurable platform models in order to find all embedded system implementations. We propose a mapping model to not only capture all implementations of a single data-flow into a single platform but to capture all data-flow variants implementations onto all platform configurations. Our variability-aware mapping model can be seen as a product line of traditional mapping models.

**Definition 3.** A variability-aware data-flow-oriented mapping \( VM = (T_m, D_m, E_m) \) where:

- \( T_m \subseteq T \times F \) is the set of possible mappings of tasks on processors \( v(t, f) \in T_m \), \( t \) can be mapped on processor function \( f \) because \( f \) can implement \( t \),
- \( D_m \subseteq D \times S \) is the set of mappings of data on memory storage,
- \( E_m \subseteq E \times (S \cup B) \) is the set of mappings of data on memory by which data are consumed/produced.

**Definition 4.** The Variability-Aware Mapping function \( M = VDG \times VRG \rightarrow VM \) sorts topologically the data-flow and finds appropriate mapping for each data, task and data-paths of the data-flow using resources of the resource graph.

Basically, each valid mapping should respect consistency constraints such as that

1. All tasks are mapped to, at least, one processor function:
   \[ \forall t \in T, \exists (t, f) \in T_m \]
2. All data are mapped to, at least, one memory storage:
   \[ \forall d \in D, \exists (d, s) \in D_m \]
3. All data-paths are mapped to, at least, one appropriate mapping. For data-path starting by an input datum, the storage on which the datum is mapped has to be reachable by the processor function on which the task consuming the datum is mapped.
   \[ \forall e = (d, p, t) \in E, \exists (e, s) \in E_m, \exists (d, s) \in D_m, \exists (t, f) \in T_m, s \in I(f) \]
   For data-path between tasks, the memory on which the output of the first task is mapped has to be reachable by the processor function on which the second task is mapped.
   \[ \forall e = (t, p, t') \in E, \exists (e, m) \in E_m, \exists (t, f) \in T_m, \exists (t', f') \in T_m, m \in O(f) \wedge m \in I(f') \]
   The mapping model of the running example \( VM_{\text{re}} \) is then formalized as
   \[ (T_m)_{\text{re}} = \{(a, a_{\text{dcu}}), (a, a_{\text{gpu}}), (b, b_{\text{gpu}}), (c, c_{\text{dcu}})\} \]

\[ D_{m_{\text{re}}} = \{(d_1, \text{RAM}), (d_1, \text{ROM}), (d_2, \text{RAM}), (d_2, \text{ROM})\}, \]
\[ E_{m_{\text{re}}} = \{(d_1, p_1, a), \text{RAM}, (d_1, p_1, a), \text{ROM}, (d_1, p_1, b), \text{RAM}, (d_1, p_1, b), \text{ROM}, (a, p_2, c), \text{r}_{\text{0}}, \text{RAM}, (a, p_2, c), \text{RAM}, (b, p_2, c), \text{RAM}, (b, p_2, c), \text{RAM}, (d_2, p_3, c), \text{RAM}, (d_2, p_3, c), \text{RAM})\} \]

Finally, The design space representing all system implementations, called variability-intensive embedded system design space is then composed of a data-flow, platform and mapping:

\[ VDS \subseteq (VDG \times VRG \times VM) \]

3.5 Design Space as a Behavioral Product Line

Automata and model-checking techniques have been widely used to model and validate real-time and embedded systems [4, 5]. Interestingly, the basic approach used is to schedule an application automaton using a platform automaton [10]. Unfortunately, these approaches are not design to manage any variability aspect of specifications.

Our framework relies on Featured-Transition-Systems (FTS) to represent and validate the design space. FTS has the strength to model explicitly the variability points structurally, through a Feature Diagram [3] (FD), instead of modeling variability points behaviorally, by optional transition with possible constraints [24]. This eases the transformation to featured automaton and the removal of invalid implementations from it. In our approach, we also use LTL property to ensure that all valid execution paths of all system implementations reach the end state of all task of the data-flow.

**Definition 5.** A featured automaton is a tuple \( FA = (\text{Loc}, \text{Loc}_0, I, \text{Act} \subseteq AF \cup \phi \cup \text{Com}, \text{trans}, \chi, \text{Ch}, L, \text{AP}, d, \lambda) \) such that:

- \( \text{Loc} \) is a finite set of locations, \( \text{Loc}_0 \subseteq \text{Loc} \), is a set of initial states and \( I \subseteq \text{Loc} \), is a set of final states,
- \( \text{Ch} \) is a finite set of communication channels,
- \( \chi \) is a finite set of variables,
- \( \text{Act} \) is a set of, \( AF \cup \phi \cup \text{Com} \)
- \( \phi \) which is a finite set of guards in a boolean expression form and \( \text{Com} \), which is a set of communications \( \text{Com} \subseteq \{e|m, e?m, e|m \in \chi} \}
- \( \text{trans} \subseteq \text{Loc} \times \text{Act} \times \text{Loc} \) are state transitions,
- \( d = (N \subseteq N_m \cup N_{opt} \cup N_{cor}, DE \subseteq N \times N, Td) \) is a Feature Diagram (FD), \( N \) is the set of mandatory, optional and alternatives features, \( DE \) represents relation between features, \( Td \) are constraints between features, \( [d]F \subseteq P(N) \) is the set of valid product configurations,
- \( \lambda : \text{trans} \rightarrow \mathbb{B}(N) \) is a total function that labels transitions with feature expressions.
- \( \text{AP} \) is a set of atomic proposition and \( L : \text{Loc} \rightarrow \text{AP} \)

A transition \( s \overset{\alpha}{\rightarrow} s' \) is possible for the set of product configurations \( P \subseteq [\lambda(s \overset{\alpha}{\rightarrow} s')] \) and if
∀g ∈ α ∩ φ, g is satisfied,
∀(c?m) ∈ α ∩ Com, wait for data event c!m,
∀(c?m0) ∈ α ∩ Com, send data event c!m0 but wait for data event c!m1 with m0 = m1.

**Definition 6.** A Linear Temporal Logic property (LTL) is a temporal expression of atomic proposition that all possible executions of system variants should satisfy as, [ϕ]FA |= ∃ where
ϕ ::= a ∈ AP | φ ∧ φ | φ ∨ φ. Symbol ∨ means that the property will become true at some point in the future.

We now show how our design space is transformed to a FA.

To simplify the transformation process, let us use the following functions:

\[ f : T ∪ Path ∪ R → N, f_o : D ∪ S × N^* → N, \]
\[ f_o : Path → N, f_{from} : Path → N, \]
\[ f_o : Path × T → N, f_{from} : T ∪ D × Path → N, \]
\[ f_m : T ∪ Path → N, f_m : T × F → N, f_{pm} : Path × S ∪ B → N, \]

transforms model elements to communication channels to interact with them at automaton level.

A first function \( Gen_{FA} : VDG → FA \times LTL \) transforms a variable data-flow graph into a FA and generates the LTL property in the following way.

1. (1.1) it transforms each datum \( d \) with variable size into a xor feature group (cf. fig. 4(a)),
   \[ ζ(d) > 1 \implies \forall s ∈ ζ(d), ∃(f(d) ∈ N_{xor}, f_s(d,s)) ∈ DE \]

2. (1.2) it creates the automaton for each source datum \( d ∈ D \) (cf. fig. 4(b)), after setting the datum size, calling the mapping automaton (cf. fig. 6(a, b)) that will allocate the datum on the memory.

   \[ ∀s ∈ ζ(d), ∃(t_o = (s_0, size(c)(m)) → s_1), \text{ where,} \]
   \[ |ζ(d)| > 1 \implies \lambda(t_o) = f_s(d,s), \]
   \[ s_1 \xrightarrow{∀p ∈ O(d), c_m(p) in} s_2, s_2 \xrightarrow{∀p ∈ O(d), c_m(p) out} s_3 \] ∈ trans

3. (1.3) it transforms each variable data-path \( p \) in a xor feature group (cf. fig. 4(a)),
   \[ |O(p)| > 1 ⇒ ∀o ∈ O(p), \]
   \[ ∃(f_{to}(p) ∈ N_{xor}, f_{to}(p,o)) ∈ DE \]
   \[ |I(p)| > 1 ⇒ ∀i ∈ I(p), \]
   \[ ∃(f_{from}(p) ∈ N_{xor}, f_{from}(i,p)) ∈ DE \]

(3.1) it creates task/data-paths consistency constraints (cf. fig. 4(a)),

\[ ∀t ∈ T, ∀p ∈ |I(t)|, |O(p)| > 1 \implies \exists(f(t) ⇔ f_{to}(p), t) ∈ T_{cl} \]

(3.2) it creates for each task \( t \) the automaton (cf. fig. 4(c)) that will wait for data-paths allocation, then call the mapping automaton (cf. fig. 6(c)) to execute the task.

(3.3) it generates the LTL formula that checks that a valid execution must, at some point, satisfy atomic proposition of all data-flow task terminal states.

\[ \varphi = (λ ∈ I, L(s) ≠ ∅) \]

**Figure 4: Partial variable data-flow application FA**

The second function \( Gen_{FA} : VRG → FA \) transforms a variable resource graph into a FA in the following way.

1. (1) it creates feature constraints on resource implementation (cf. fig. 5(a)).
∀r ∈ R, θ(r) = T → ∃f(r) ∈ N_{opt}
∀r ∈ R, ∀req ∈ φ_{requires}(r), ∃(f(r) → f(req)) ∈ Tcl
∀r ∈ R, ∀exec ∈ φ_{executes}(r), ∃(f(r) → −f(exec)) ∈ Tcl

(2.1) it creates features representing all possible data-path mappings on memory.
∀((x, p, y), s) ∈ Em, ∃(f_m(p) ∈ N_{xor}, f_pom(p, s)) ∈ DE

(2.2) it creates for each data-path mapping the automaton that allocates memory (cf. fig.6(b)).
∀p ∈ Path, ∀((x, p, y), s) ∈ Em,
∃{s_0 \xrightarrow{c_m(p, s)} out_s_1, t_0 = (s_1 \xrightarrow{c(p, s)} out, loc(d) = s_2)}
where, λ(t_0) = f_pom(p, s), s_2 \xrightarrow{c_m(p, s, c)} \in trans

Finally the function \text{Gen}_{FA} : \text{VDS} → \text{FA}, defined by:
\text{Gen}_{FA}((\text{vdg}, \text{vrg}, \text{vm})) :
\text{Gen}_{FA}(\text{vdg})||\text{Gen}_{FA}(\text{vrg})||\text{Gen}_{FA}(\text{vm}),
transforms our design space into a featured automaton.

A third function \text{Gen}_{FA} : \text{VM} → \text{FA} transforms a variability-aware dataflow-oriented mapping into a FA as follows.

(1.1) it creates features representing all possible task mappings on processor function (cf. fig. 6(a)).
∀(t, f) ∈ Tm, ∃(f_m(t) ∈ N_{xor}, f_m(t, f)) ∈ DE

(1.2) it creates for each task mapping the automaton that executes the processor function according to the mapping configuration (cf. fig. 6(c)).
∀t ∈ T, ∀(t, f) ∈ Tm, ∃{t_0 = (s_0 \xrightarrow{c_m(t, s)} out_s_1), where,
\text{f_m}(t) ∈ N_{opt} \implies λ(t_0) = f_m(t) ∧ λ((s_0 → s_3) ∈ trans) = −f_m(t)
\text{t_1} = (s_1 \xrightarrow{c_f(t, s)} out_s_2), where, λ(t_1) = f_m(t, f),
\text{s_2} \xrightarrow{c_m(t, s)} out_s_3}, ∈ trans

Figure 5: Partial variable platform FA
To preserve the consistency of the design space, variability constraints are inferred such as:

(1.1) Task features with variable data-path features are not implemented on all data-flow variants, then such features are made optional (cf. fig. 4(a)).

\[ \forall t \in T, \exists p_t \in I(t), [O(p_t)] > 1 \lor \exists p_o \in O(t), [I(p_o)] > 1 \implies f(t) \in N_{opt} \]

(1.2) Variable task features have their mapping variable too; if a task feature is implemented its mapping must be implemented too, and vice-versa (cf. fig. 4(a) & 6(a)).

\[ \forall t \in T, f(t) \in N_{opt} \implies f_m(t) \in N_{opt} \land (f(t) \iff f_m(t)) \in TcI \]

(2.1) If a task mapping feature is implemented on a processor function, the implemented input and output path mappings have to be reachable (cf. fig. 6(a)).

\[ \forall (t,f) \in Tm, \forall p_t \in I(t), \forall p_o \in O(t), \exists f_m(t,f) \iff (f(t) \implies f_m(t)) \in TcI \]

(3.1) If a task mapping feature using an optional processor is implemented, the processor must be implemented too.

\[ \forall p = (F,x,y) \in Proc, f(p) \in N_{opt}, \forall (t,f) \in F \implies f_m(t,f) \land f_m(t) \implies f(p) \in TcI \]

Similarly, if a data-path mapping feature is implemented on fifo buffer of optional processor (3.2) or optional memory storage (3.3), the resource have to be implemented.

(3.2) \forall p_u = (F,B,x) \in Proc, \forall (y,p,z), b \in B \in Em, f(p_u) \in N_{opt} \implies f_m(p_u,b) \implies f(p_u) \in TcI

(3.3) \forall (x,y,p), s \in S \in Em, f(s) \in N_{opt} \implies f_m(p,s) \iff f(s) \in TcI

As an illustration, in our running example, the rules would be:

(1.2) \( A \iff A_m, B \iff B_m \)

(3.1) \( BOnB_{gpu} \implies GPU, AOnA_{gpu} \implies GPU \)

(3.2) \( P2OnR0_{gpu} \implies GPU \)

(3.3) \( P1OnRAM \implies RAM, P2OnRAM \implies RAM \)

\( P3OnRam \implies RAM \)

3.6 Validation Process

As our form of behavioral product lines is based on FTS [8], model checking techniques can be directly reused. In our implementation (cf. next section), we use the ProVeLines checker as a back-end for the validation process. The process consists in verifying all execution paths of all products \( [f_a]_{PA} \) of the product line, in an efficient way by exploiting commonalities between different products. Theoretically, more the products share common behavior, and the more efficient should be the variability aware model checking in comparison of iterative model checking on individual systems [7]. Instead of exploring all executions for each system implementation, the model checker explores an execution \( \pi \) once for all implementations \( P \) able to produce this specific execution:

\[ P = \{ p \in [d]_{FD} | \pi \in [f_a]_A \} \]

As mentioned in the previous section, some system configurations may expose inconsistent behaviors (e.g., memory allocation error, violation of graphical pipeline constraints). These behaviors will abort the execution and the basic properties (e.g., safety, absence of deadlock, state reachability) will obviously not be satisfied. In our validation process, we are able to remove these configurations from the system by relying again on the back-end model checker [7]. It computes the set of bad product configurations, which we remove from the feature diagram of the product line by adding the appropriate cross-tree constraints.

4. VALIDATION

4.1 Implementation

4.1.1 Overview

The framework depicted in Fig. 3 has been entirely implemented in Java. It consists of 3 main modules: i) meta-models of variable application and configurable platform (cf. Fig.7 and 8) ii) mapping meta-model Fig.9 and algorithm (cf. listing 3) iii) generators that transform the design space...
composed by all system sub-domains (application, platform, mapping) (cf. listing 4) into formal models of behavioral product line (cf. listings 5 and 6) in order to remove invalid products (cf. listing 8) reusing automated formal reasoning techniques (cf. listing 7).

The first module allows for specifying a variable data-flow oriented application (cf. listing 1) and a configurable platform (cf. listing 2) via fluent APIs. As a result, the running example inputs are captured in less than 30 lines of code. The second module calls our mapping algorithm (cf. listing 3) in order to infer, at the end, the resulting design space. The third and last module transforms the design space into a Feature Model in TVL (cf. listing 5) and a Featured Automaton in fPromela (cf. listing 6), capturing, respectively, the structural variability and behavior of the design space.

We reuse the ProVeLines model-checker [9, 6], which consumes TVL and fPromela inputs to assess all system designs, in one run of variability-aware model checking. The resulting outputs, printed as a set of invalid sub-products lines (c.f. listing 7), are directly used to constraint the design space to only obtain valid products (cf. listing 8).

4.1.2 Applications as Variable Data-Flows
Listing 1 illustrates how we capture the functional requirements (cf. fig. 1) of the embedded system through an extended data-flow Java API. The data-flow meta-model (cf. Fig. 7) contains the classic structure and behavior of the data-flow (data instanced at line 3,6, task at line 4,5,7, data-path at line 2), but also captures the variability in both structural properties (e.g., data size at line 6) and behavioral properties (e.g., alternative flows by allowing data-paths to have multiple input and output tasks connected at line 4,5).

Listing 1: Running Example Application

```
Application app = new Application("WarpWithWhat");
Path p1 = app.addPath("P1"); Path p2 = ...; Path p3 = ...
DataSource d1 = app.addDataSource("D1").addSize(512).connect("o", p1);
Task ta = app.addTask("ta", "A").connect(p1, "i").connect("o", p2);
Task tb = app.addTask("tb", "B").connect(p1, "i").connect("o", p2);
DataSource d2 = app.addDataSource("D2").addSizes(512, 1024).connect("o", p3);
Task tc = app.addTask("tc", "C").connect(p2, "i0").connect(p3, "i1");
app.split(p1).to(ta).to(tb);
app.join(p2).from(ta).from(tb);
```

4.1.3 Platforms as Variable Resource Graphs
Listing 2 illustrates how we express the platform specification (cf. fig. 2) of the embedded system through a resource component based Java API. The platform meta-model (cf. Fig. 8) contains templated resource components such as multi-pass processors instanced at line 13 and streaming processor at line 6. Other elements in the template can be hardware functions instanced at line 7,10,14, read-only memory at line 2, read-write RAM memory at line 3, first-in-first-out buffers at line 8,14, relevant elements being connected with each others. In addition, a platform can have optional resource components (line 4,13) and variability dependency (line 15) on resources.

Listing 2: Running Example Platform

```
Platform plt = new Platform("Kepler");
Storage rom = plt.addStorage("ROM", Type.READ_ONLY).addCapacity(4096);
Storage ram = plt.addStorage("RAM", Type.READ_AND_WRITE).addCapacities(1024, 4096);
ram.setOptional();
Component dcu = plt.addComponent("DCU");
Processor a_dcu = dcu.addProcessor("a", "A");
Memory r0_dcu = dcu.addFIFOBuffer("R0");
a_dcu.connectToInputPort("i", ram, rom).connectToOutputPort("o", r0_dcu);
Processor c_dcu = dcu.addProcessor("c", "C");
c_dcu.connectToInputPort("i0", ram, rom, r0_dcu).connect("i1", ram, rom, r0_dcu);
Component gpu = plt.addComponent("GPU").setOptional();
Processor a_gpu = ...;Memory r0_gpu = ...;Processor b_gpu = ...;
gpu.requires(ram);
```

4.1.4 Variability-Aware Mapping Process
The mapping algorithm (cf. listing 3) takes as inputs the variable data-flow and configurable platform Java models, and generates the Variability-Aware Mapping Space (cf. Fig. 9 for metamodel). It then represents all mapping of application elements onto platform resources.
The process is composed of two steps: (i) it maps each data source and output data path on storage memories (line 4-7) (ii) it maps each task on appropriate processor function (i.e., processor function can implement the task while data path inputs can be mapped on reachable memory) and maps task output on memory (line 8-11). Then, the algorithm prunes unfeasible mappings w.r.t. structural and variability constraints at line 12 (e.g., data-path mapping are not reachable by any task mapping or vice versa), finally adding appropriate constraints to ensure mapping space consistency (line 13).

4.1.5 Design Space as a Behavioral Product Line

From the system design space model (cf.listing 4), which is the consistent composition of our 3 system sub-domains (i.e., application, mapping, platform), a Behavioral Product Line representing all system implementations is generated into a Feature Automaton (cf.listing 4). While the behavior of the whole product line is encoded in fPromela (cf. listing 6), a Feature Model in TVL encodes its structural variability (cf.listing 5).

Our framework relies on Featured-Transition-Systems (FTS) to formally reason on the structure and behavior of the design space. FTS has the strength to model explicitly the variability points structurally, through a Feature Diagram [3] (FD) in TVL (cf.listing 5).

Thus, a single variability model is able to capture data size at line 9-12, memory capacity at line 19-22, data mapping 28-31, task mapping 32-35, alternative flow 9-12,13,14 variabilities, resource optionality 18,24, resource dependency 45, mapping 41 and design space consistency constraints 39, 43. On the other hand, the behavior of the design space is captured by an executable network of featured automata in fPromela where state transitions are guarded by constraints on feature values.

To execute the variable application over the configurable platform, featured automata that capture behavior of data-flow processes, such as data node (lines 18-34) and task node (lines 36-52), call functions over platform resource featured automata, such as memory storage (line 2-16) and processor.

Each featured automaton may have variable properties such as capacity for memory storage process (lines 8-11), data size (lines 21-24) and data deployment location (lines 25-30) for data node process, or task deployment (line 42-47) for task process. In addition to properties, a featured automaton may be optional (i.e., behavior is executed if the element is present in the system design cf. lines 39-50).

According to the subset of design decisions to explore, variable properties are incrementally fixed. For example, for system designs where D2 with a size of 1024 is deployed on RAM with a capacity of 1024, we observe that after allocating D2 on RAM (line 26,6), RAM is full, and any other data allocation on RAM (e.g., D1 or data of P2) would lead to a memory violation (cf. listing 7 line 4).
... active proctype Storage_RAM(){ atomic{
  Data in;
  ... do
  :: RAMalloc?in ->
    cons = cons + in.size;
  if
  :: RAM_capacity_1024 -> size = 1024;
  :: RAM_capacity_2048 -> size = 2048;
  fi;
  assert(cons <= size);
  RAMalloc!in;
  od;
...}
}

active proctype Data_D2(){ atomic{
  Data out;
  ... if
  :: D2_size_512 -> out.size = 512;
  :: D2_size_1024 -> out.size = 1024;
  fi;
  if
  :: D2_On_RAM -> RAMalloc!out;
    RAMalloc?eval(out);
  :: D2_On_ROM -> ROMalloc!out;
    ROMalloc?eval(out);
  fi;
  ... P3!out;
...}
}

active proctype Task_TA(){ atomic{
  Data in, out;
  ... if
  :: TA -> P1?in
  ... if
  :: TA_On_GPU_a -> GPU_a!in, out;
  ... :: TA_On_DCU_a -> DCU_a!in, out;
  fi;
...}

4.1.6 Validation Process

The generated formal models (fPromela and TVL) are checked through ProVeLines with specific command lines (cf. listing 7 line 1). It returns output containing non-feasible subsets of products (line 4) that are used to invalidate variants by constraining the design space variability space (cf. listing 8 line 7). More precisely, the model checking is going to verify products against inconsistent behaviors (e.g., memory allocation error, violation of graphical pipeline constraints) and more classic properties, such as safety, absence of deadlock and state reachability on all variants in one run. This single execution [7] makes it also possible to remove all products (cf. listing 7 at line 4) leading to an invalid execution, so to improve and speed-up the verification process (see next Section).

4.2 Evaluation

4.2.1 Industrial Use Case

In order to validate our toolled approach on an industrial scale, we applied it to a real low-end market instrument cluster provided by Visteon, the automotive systems company we collaborate with.

The functional requirements of the cluster represent a variable data-flow with 3 source images processed by 8 tasks connected by 9 data-paths. Each source image has two different resolutions (i.e., HD and LD) and two tasks sub flow sequences are alternative through a xor join/split data-path.
resulting in 16 data-flow variants. The platform specification of the cluster is then represented by a variable hardware component system with 2 memories (a Video RAM and a ROM Flash) and 3 processors (two multi-pass GPU bitblitter and one streaming-based DCU). Each processor has a pipeline processing of 4 stages and 3 fifo buffers. In terms of platform variability, the 2 bitblitters and the VRAM are optional. Each memory has 2 different configurable sizes at manufacturing time. The number of platform configurations in the use case is then 24.

If each data-flow variant had one possible implementation on each platform configuration, the number of different cluster system implementations would be 384. In reality, some platform configurations do not provide the graphical functionalities required by some data-flow variants. Furthermore, due to multiple task implementation choices, data-flow variants have several thousand possible implementation alternatives onto a platform configuration. Setting the platform configuration to the higher end (i.e. selecting VRAM and all processors), one can find 72 and 78 possible implementations of two data-flow variants that take different xor data-path decisions. This is due to more pipelining opportunities in the second data-flow variant, even if there is more data-path mapping possibilities in the first one.

Table 1 shows time measurements of the complete toolchain while varying the different variability dimensions over the use case. In the first seven rows, we observe that the whole process is performing well with small to medium scale of variabilities. Data and memory size variability verifications are faster and require more state exploration than platform component and data-path variabilities. Component and data-path variabilities are also slower to check than data and memory size variabilities. It is likely to be due to the fact that contrary to size variability, hardware component and data-path variability are strongly impacting the implementation variability, and consequently the state space of the model checker.

We have also complemented this experimentation by taking a single structural data-flow from the industrial use case with a simulated larger platform, itself with multiple memories and processors. Results in the last three rows of Table 1 show that solving can scale to a large number of implementation variants. Even if the solving time is significant, we observe that the number of states explored to assess all the implementation variants is significantly low. This shows that behavioral commonalities between system variants are used to speed-up the verification process.

### 4.2.2 Scalability of System Variabilities

We now analyze the scalability of our solution against system variability by increasing the application, platform and mapping variability on simulated data. Fig. 10.a shows measurements of our toolchain for data size and memory capacities variability dimensions (called element variability) while Fig. 10.b is about mapping dimensions. Each variable system denotes his variability dimensions in the following format:

\[ \text{flow; size; resource; capacity; mapping} \]

Where flow represents the sum of flow variants of variable data-paths, size the sum of alternatives data sizes, resource the number of optional resources, capacity the sum of alternatives memory capacities and mapping the sum of alternatives mapping of application element onto platform resources.

Fig. 10.a shows a system where data size and memory capacities variability dimensions have been progressively increased so that the first and last system counts, respectively, 384 and 6912 variants. We show relative time and (automaton) states metrics – in total and per variants – compared to the normalized system presenting the lowest variability. Thus, for element variability dimensions (data size and memory capacity), even if, obviously, the time and states number needed to verify the system increase according to variability dimensions, the verification time and states number needed by variant decrease.

For the scalability of the mapping variability dimension, we mainly increase progressively the mapping dimension over 5 systems, the lowest system containing 64 variants while the highest has 23328. We observe that the needed time and explored states number grow quickly. We think this is due to the intrinsic high complexity of both binding and scheduling [22], which leads to configuration space and state space explosion during checking. However, the verification speed-up by variant is still interesting for high variability system.

<table>
<thead>
<tr>
<th>Variability</th>
<th>Implementation variants</th>
<th>Platform variants</th>
<th>data-flow variants</th>
<th>States explored (re-explored)</th>
<th>Time (ms)</th>
<th>ms / variants</th>
<th>states / variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>78</td>
<td>0</td>
<td>0</td>
<td>2453 (331)</td>
<td>27</td>
<td>0.346</td>
<td>31.448</td>
</tr>
<tr>
<td>Data size</td>
<td>624</td>
<td>0</td>
<td>8</td>
<td>15254 (2406)</td>
<td>201</td>
<td>0.322</td>
<td>35.976</td>
</tr>
<tr>
<td>Platform</td>
<td>424</td>
<td>24</td>
<td>0</td>
<td>5673 (546)</td>
<td>65</td>
<td>0.153</td>
<td>13.379</td>
</tr>
<tr>
<td>Platform and data size</td>
<td>3392</td>
<td>24</td>
<td>8</td>
<td>37435 (4856)</td>
<td>602</td>
<td>0.177</td>
<td>11.036</td>
</tr>
<tr>
<td>data-path</td>
<td>150</td>
<td>0</td>
<td>2</td>
<td>4727 (981)</td>
<td>74</td>
<td>0.493</td>
<td>31.513</td>
</tr>
<tr>
<td>data-path and data size</td>
<td>1200</td>
<td>0</td>
<td>16</td>
<td>29066 (6994)</td>
<td>587</td>
<td>0.489</td>
<td>24.222</td>
</tr>
<tr>
<td>ALL</td>
<td>&gt;4800</td>
<td>24</td>
<td>16</td>
<td>72704 (14652)</td>
<td>2361</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Platform mult. mem</td>
<td>16408</td>
<td>40</td>
<td>0</td>
<td>134941 (4534)</td>
<td>4010</td>
<td>0.244</td>
<td>8.224</td>
</tr>
<tr>
<td>Platform mult. proc</td>
<td>2848</td>
<td>80</td>
<td>0</td>
<td>19625 (3224)</td>
<td>337</td>
<td>0.118</td>
<td>6.890</td>
</tr>
<tr>
<td>Pltf. mult. proc &amp; mem</td>
<td>516608</td>
<td>160</td>
<td>0</td>
<td>1341999 (175304)</td>
<td>289721</td>
<td>0.560</td>
<td>2.597</td>
</tr>
</tbody>
</table>
Figure 10: Variability Analysis

(a) Element Variability

(b) Mapping Variability

Figure 11: Complexity Analysis

(a) Mapping Complexity

(b) Flow Complexity
4.2.3 Scalability of System Complexities

We analyze here the scalability of our solution against system complexity by growing the application and platform size. Fig. 11.a shows an example where we increased progressively the size and complexity of both application and platform sides while keeping the mapping variability at a common factor (i.e., $\%40$).

The normalized system presenting the lowest complexity and variability. It contains 14 application nodes and paths mapped on a 10 resources platform with 48 variants, while the most complex system has 22 nodes, 13 resources, and 2880 variants. A global system complexity metric, taking into account the system size and its number of variants, has been discussed with our industrial partner.

We observe that the verification time and states needed increase quickly according to the system complexity (application and platform sizes). Interestingly, as the needed states and time to explore per variant in more complex while the system is growing, the verification time per variant is still lower. This observation can also be made on Fig. 11.b, where we progressively increase the complexity of a system with flow variability.

4.3 Threats to Validity

The validation on a single case study can be considered as the major internal threat to validity. Nonetheless it was incrementally built by numerous meetings with different domain experts of the automotive systems company. We gathered specification and feedback from several applications and platforms case studies, and we finally chose the presented one as the most representative among them.

As for external threats, we identified the data sets as the main issue. While the data sets are large, they are still simulated. Our creation procedure has been built to mimic the structure and behavior of both the platforms and the applications, taking the real case studies as a basis to be expanded by generators. Still we do not have empirical evidence of their correspondence.

5. CONCLUSION AND FUTURE WORKS

A tremendous amount of variability can be observed in embedded systems, and especially in data-flow oriented ones, which are now systematically built from highly variable specifications and target diverse hardware platforms configurable at a very high level of detail. To handle the early functional assessment of all these possible configurations, we proposed in this paper a tooled approach that takes variable data-flow specifications and variable hardware platform models to map them together and transform them into a behavioral product line representing the potential design space. These models and toochain allow to use automated reasoning techniques to explore and assess the functional feasibility of all represented variants in a single run, and invalid products can be removed by adding constraints to the product line.

We reported on the application of the proposed approach to a real-world industrial use case of automotive instrument cluster, giving hints on a potential good applicability. Our experimental validation with large simulated datasets also shows a good scalability of the prototype implementation for industrial-scale applications and platforms.

As future work, we first plan to facilitate the usage of the framework with domain specific languages for input models (specification and platform), and to conduct larger experiments with them on different and new case studies from our industrial partner. Interestingly, we think that some product lines optimization techniques [11, 23] could be applied to assess more variable and complex embedded systems.

We will also extend our variability-focused framework by taking into account quality attributes (e.g. cost, run-time etc.). The extension would then provide optimized product selection as a complement to the functional validation presented in this paper. We expect this more complete framework to be applicable in different contexts, being similar in the separation of application models being mapped onto component-based platforms.

6. ACKNOWLEDGMENTS

We thank Visteon Electronics and the Association Nationale de la Recherche et de la Technologie for continuously supporting this research. We thank also Emmanuel Roncoroni and Olivier Bantiche who brought industrial expertise in instrument cluster engineering and Maxime Cordy for his valuable support on the ProVeLines model-checker.

7. REFERENCES


ABOUT THE AUTHORS:

Sami Lazreg is a research engineer and a PhD candidate at Visteon Electronics (US Company) and Université Côte d’Azur (France) where he received his MSc in Computer Science with honors. He started as an apprentice for Visteon Company and became a technical expert of software and system engineering focusing 3D rendering capabilities for low-end and mid-end instrument clusters.

Philippe Collet is a professor at Université Côte d'Azur (France). He is a deputy director in charge of corporate relations at Polytech'Nice Sophia, the MSc level engineering school of Sophia Antipolis, and is also coordinating the apprenticeship system of the Computer Science MSc. His research interests cover software product line engineering and variability management.

Sébastien Mosser is an associate professor at Université Côte d'Azur (France). He coordinates the Software Architecture specialization of the Computer Science MSc, and teaches software engineering courses at both graduate and undergraduate levels. His research interests cover software development and separation of concerns mechanisms, with a particular focus on domain-specific languages, model-driven engineering and software composition.
An Effective Sequence Structure Representation for Long Non-Coding RNA Identification and Cancer Association using Machine Learning Methods

Manu Madhavan  
Dept of Computer Science and Engineering  
National Institute of Technology Calicut  
Kerala, India- 673601  
amu_p150091cs@nitc.ac.in

Gopakumar Gopalakrishnan Nair  
Dept of Computer Science and Engineering  
National Institute of Technology Calicut  
Kerala, India- 673601  
gopakumarg@nitc.ac.in

ABSTRACT

The invention of high-throughput technologies and consequent developments in Bioinformatics research unveiled many important non-coding transcript molecules such as Long non-coding RNAs (lncRNAs). The available studies confirmed that lncRNAs play important genetic and epigenetic roles in higher-order species like the human and their differential expressions lead to complex diseases like cancer. Even though there are arrays of studies and related tools for the analysis, less conserved patterns in the sequences and intractable structural properties challenge the understanding of varying functionalities of lncRNAs. For the better approximation of these characteristics, higher quality feature representation is required. This paper proposes an extended hybrid sequence-structure feature set for machine learning based lncRNA analysis. Here, the sequence features are derived from various frequencies of k-mer patterns, GC content and molecular weight. The structure representations consider the context of different secondary structure elements which include stems, interior loops, multi-loops and hairpin loops. These features are used for the classification of lncRNA/mRNA and cancerous/non-cancerous lncRNAs. The classifications use machine learning algorithms such as LDA based topic model, Random Forest, SVM and Naïve Bayes. The results show that the proposed feature set is effective in classifying lncRNAs and provide a direction towards the analysis of the role of secondary structure elements in cancer-related lncRNAs.

CCS Concepts

• Applied computing → Bioinformatics; • Information systems → Data mining;

Keywords

Long Non-Coding RNAs; RNA Secondary Structure; Feature Selection; Cancerous lncRNAs, Latent Dirichlet Allocation; Support Vector Machine; Random Forest; Naïve Bayes

1. INTRODUCTION

The central dogma of molecular biology explains RNAs as an intermediary between DNA and proteins. With the advancement of transcriptome sequencing techniques such as RNA-seq, numerous transcripts have been identified. Most of them do not translate into proteins but play a significant role in the synthesis and regulation of translation activities. Such transcripts are commonly called non-coding RNAs (ncRNAs), and long non-coding RNAs (lncRNAs) are a particular kind of ncRNAs. By definition, lncRNAs are non-protein coding transcripts with sequence longer than 200 nucleotides. The recent developments in biotechnology research have identified thousands of new lncRNAs, found to be up-regulated or down-regulated in the molecular mechanism [19]. It is reported that lncRNAs control many important biological processes and any dysregulation will lead to complex diseases like cancer, cardiovascular disease, and neuro-degeneration disease [23].

Though lncRNAs like XIST, HOTAIR, ANRIL etc. are well studied, functionalities of a majority of lncRNAs are mostly unknown. Identification and functional analysis of lncRNAs are crucial for the understanding of genetic and epigenetic activities and development of diseases. Lack of sequence conservation and tissue-specific low-level expressions cause the analysis of lncRNAs challenging [10].

The main activities involved in the computational studies of lncRNAs are identification, analysis and functional annotation. The identification of lncRNA from a collection of uncharacterized transcript sequence is the prime and crucial step in these analyses. The machine learning based algorithms [22, 28] have been introduced by the researchers with different efficacy. The power of machine learning algorithms relies on the coherent feature extraction from the input data collection. Most of the recent machine learning models for lncRNA analysis use information from the sequence such as k-mer pattern, GC Content and ORF length [26, 29, 30]. Unfortunately, general studies [19] on lncRNAs show that they are less conserved in sequence level than the secondary structure, which shows more conserved patterns.

RNA secondary structure is formed through the formation of base pairs (G-U, A-U and G-C) between runs of self complementary sequences in RNA molecule. The paired base regions constitute the stem and unpaired regions folded into various loops in the secondary structure. The stems provide structural stability to RNA molecule while the differ-
ent loops contribute to the various molecular interactions and hence the functionalities of RNA [21]. For instance, some studies [25] have found that folds and loops in IncRNAs such as MALAT1, HOTAIR, and SRA act as conserved motifs for the function identification. Though the structure-function correlation in IncRNAs is typical, there is a large number of exception showing un-conserved structure motifs [6]. The available methods involving structure features limited to minimum field energy (MFE) and some simple vectorization of paired and unpaired nucleotide frequencies [18]. A meaningful representation considering the presence of various structural elements is essential for the structure based analysis of IncRNAs.

Coming to the various machine learning models for transcript analysis, one approach considers biological sequences as text and applies techniques inspired by natural language processing. In Bioinformatics, topic models such as Latent Dirichlet Allocation (LDA) are used to model biological objects and molecular activities [16]. Other machine learning techniques like Random Forest (RF), Support Vector Machine (SVM) and Naïve Bayes are used extensively in recent tools [11, 13, 29] for IncRNA analysis.

In this paper, we present systematic identification and representation of sequence-structure features for the analysis of IncRNAs. The principal goal of the current study is the identification of hybrid sequence-structure feature set for IncRNAs. The major tasks included in this work are the vectorization of transcript sequences using enhanced k-mer representation and development of vital secondary structure feature quantization methods utilizing the knowledge of various structural elements. The experiments use the proposed features for two applications- IncRNA/mRNA classification and cancerous/non-cancerous IncRNA classification. The classification uses machine learning algorithms such as LDA, RF and SVM. The features discussed are new extensions of previous works available in the literature and results show that the proposed features are significant. The study also overlooks the importance of proposed features in distinguishing cancer-related IncRNAs and presents statistical reasoning of the role of structural elements in cancerous IncRNAs, with biological evidence.

Rest of the paper is organized as follows. Section 2 discuss the major works related to this field of study. Section 3 outline the methodology and the feature set combinations proposed in this work. Results and important observations are listed in Section 4 and Section 5 conclude the paper by giving future direction in this area.

2. RELATED WORK

Recently, several machine learning works have been developed for the analysis of IncRNAs. For this study, we organized them into two groups- sequence-based methods, structure based methods. Most of the computational works related to IncRNA analysis use primary sequence derived features. PLEK [13], Coding Potential Calculator (CPC), and PhyloCSF are some of the tools available for the prediction of (long)non-coding RNAs showing different efficiencies [10, 28]. CPC is a support vector machine based classifier for identifying non-coding transcripts, which uses ORF and local alignment score as features. PLEK- predictor of IncRNA and messenger RNAs based on improved k-mer scheme [13] is an alignment-free method using the combination of k-mers (k=1 to 5) as features and SVM classifier. IncRScan-SVM [29], IncRNA-ID, and LncRNA2Function [11] are some of the recent studies on IncRNA analysis, rooted on machine learning and sequence features. These works use k-mer distribution as a feature along with other chemical properties of the RNA molecule.

The study of [25] concludes that secondary structure of IncRNAs plays crucial roles in their functionality. However, the way in which sequence and structure affect various functionalities is largely unknown. Ventola et al.,[30] presented a systematic feature selection and comparative study for the analysis of IncRNA transcripts. They collected 125 genomic features, depending on the species and grouped into five categories: basic features, open reading frame (ORF), conservation score, nucleotide compositions and repeated elements. The tools like COME [9] and nRC [7] use secondary structure for IncRNA classification. The tool nRC [7] represented RNA secondary structure as a graph and adopted Molecul Substructure (MoSS) algorithm to identify the frequent subgraphs in the secondary structure. Dogan et al.,[6], proposed a feature generation algorithm to generate structure-based features from position specific paired k-mer of the RNA string. The tool repRNA [15] provides methods for various sequence-structure feature representations for RNA.

3. MATERIALS AND METHOD

The primary focus of the current work is the identification of significant features for IncRNA studies. Based on the previous studies [10, 31] it is clear that IncRNAs show a little sequence level conservation. In addition to this, [25] revealed that the structure level conservation of IncRNAs is also not prominent as in coding RNAs or proteins. In this juncture, we assume that a hybrid of sequence-structure features serves better than the individual categories of features. In this current study, we used various combinations of sequence-structure features to classify IncRNAs and coding RNAs. The sequence features include k-mer combination, GC content and molecular weight derived from the nucleotide sequence. The structure features include minimum free energy (MFE), structural fragment sequence (Frag_seq) and structure status component (SSComponent). For the purpose of machine learning based studies, each IncRNA or mRNA sequence is considered as a document and represented as bag-of-feature words. The effectiveness of the feature combination is measured by applying in a classification task. The following subsections give the detailed explanations of each feature.

3.1 Sequence Features

The sequence level features are derived from the primary nucleotide sequences of the IncRNA or mRNA. The primary sequence is considered as a document coming from the alphabets $\sum = \{A, C, U, G\}$. The sequence level features are obtained from the normalized frequency of various substring patterns of the sequence.
3.1.1 k-mer combination

k-mers are all the possible substrings of length k that are contained in the sequence. k-mers act as words in the documents. The idea of using k-mer frequency as feature is based on the works [13, 29]. These works use all possible k-mers (1 ≤ k ≤ 5) as feature words, which leads to a feature space of 5000 k-mers. It is observed that the frequency and significance of k-mers are not equally likely. Hence, instead of directly using frequency of k-mers, we rank the k-mers based on the term-frequency inverse document frequency (tf-idf) weights. TF-IDF ranking is a concept taken from information retrieval. Term frequency is the normalized frequency of a word in the document. Inverse document frequency is the log-normalized measure of number of documents in which the particular k-mer appear. The weight assigned to a k-mer is the product of tf and idf (say weight of k-mer k\textsubscript{i} in sequence S\textsubscript{j} is, w\textsubscript{ij} = tf\textsubscript{ij} \times idf\textsubscript{1} [4]). The earlier work [20] observed that only top 15 k-mers, for 3 ≤ k ≤ 7, positively contribute to characterize the lncRNAs or mRNAs. The current work combine all top 15 k-mers of 3 ≤ k ≤ 7 and this combined k-mer set is used as feature for topic modeling.

3.1.2 GC content

GC Content is the percentage of nitrogenous bases on a DNA or RNA molecule that are either guanine(G) or cytosine(C) from a possibility of four different ones (G, C, Adenine(A), and Uracil(U)). It is calculated as the portion of G and C in the total sequence (see equation 1) [30]. We consider the GC content of whole sequence. Since the GC pair is bounded by three hydrogen bonds and AU pair is bounded by two hydrogen bonds, high GC content provides high stability to the molecular structure [21]. Recent transcript analysis has revealed that the GC content of IncRNA is low compared to that of coding RNAs [22].

\[ GC_{\text{Content}} = \frac{C(G) + C(C)}{C(A) + C(C) + C(G) + C(U)} \]  
(1)

where C(x) represents the count of nucleotide base \( x \in \{A, C, G, U\} \).

3.1.3 Molecular weight

Molecular weight is the mass of a molecule, calculated as the sum of the atomic weights of each constituent element multiplied by the number of atoms of that element in the molecular formula [21]. The use of molecular weight as a feature is motivated by the important observation that IncRNA have high molecular weight compared to mRNAs [14]. We used Biopython [5] package for computing molecular weight of the sequence.

3.2 Structure Features

Secondary structure may be key for the function of some of the lncRNAs, supported by several studies on functional analysis of lncRNAs. Some studies have found that specific structural elements like loops and folds are conserved in species level [25]. Minimum free energy (MFE) is the standard secondary structure feature used in RNA analysis. Recent works such as [15] considered various structural elements such as stems, hairpin, bulges, and internal loops in the secondary structure. The critical challenge in structural feature extraction is effective quantization and vector representation. In this work, we propose two such representations for RNA secondary structure.

3.2.1 Secondary structure status component

Secondary structure status components (SSComponent) of the RNA are base pairs and unpaired bases. The base pairs form the stem region and provide stability to the structure, whereas unpaired regions folded into different loops of the secondary structure. There are six possible base pairs in the RNA secondary structure and four unpaired nucleotide bases. Hence, given RNA sequence, a 10-dimensional vector, SSComponent = \{A, C, U, G, A−U, U−A, G−C, C−G, G−U, U−G\}, can be computed, where each \( x \in SSComponent \) is the normalized frequency of unpaired bases and each \( x \in SSComponent \) is the normalized frequency of base pairs. The above representation is a simple modification of representation proposed in [15]. RNAfold [17] algorithm is used for the extraction of secondary structure from the sequence, (in dot-bracket notation) and Forgi 1.1 [12] is used for the base pair identification.

3.2.2 Secondary structure fragment sequence

The fragbag [3] concept applied for protein structure motivates the fragment sequence(Frag_seq) representation of RNA secondary structure. Here, we consider RNA secondary structure as a composition of 4 types of structural elements- stem, interior loop (bulges), hairpin loop and multi-loop. The sequence of nucleotide bases forming each structural element can be considered as a fragment (or word). With this bag-of-structural fragments representation, secondary structure can be considered as a short-document and fragment sequences as words. The topic distribution with the number of topics=4) gives the distribution of structural elements. Since the distribution of these structural elements is not abundant in all input sequences, the count vectorizer will give a sparse vector. So, we normalize the count vector with soft-max normalization before applying LDA.

3.2.3 Minimum Free Energy

We calculate the minimum free energy (MFE) of RNA sequence using RNAfold [17] package. Since the MFE of RNAs increases with sequence length, we normalize the feature values by dividing the value with the number of nucleotides. This value provides the folding stability of RNAs of various sizes.

3.3 Experiment Design

The features designed in the previous subsections are used for two binary classification tasks- lncRNA/mRNA and cancerous and non-cancerous lncRNA classification. The input is a set of lncRNAs and mRNA sequences with class labels. In LDA based classification, the words in the feature space are mapped to topic space, where word distribution defines each topic. This work assume that assume that each document belongs to one topic, that determines the class of the document. LDA provides topic distribution, not class labels, and we use the RF algorithm for the label prediction. For other algorithms, each transcript sequence is represented as a vector of selected features.
Figure 1: Comparison of accuracy of 4 machine learning algorithms using different structure features

Figure 2: lncRNA/mRNA classification: Comparison of accuracy and f-score of 4 machine learning algorithms for different feature combinations with GENCODE dataset

Table 1: Description of Data set used

<table>
<thead>
<tr>
<th>Task</th>
<th>Data Set</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>lncRNA-mRNA</td>
<td>GENCODE v27</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>Cancerous lncRNA</td>
<td>Lnc2cancer database</td>
<td>150</td>
<td>150</td>
</tr>
</tbody>
</table>

3.4 Dataset

The method is tested on two different datasets. For testing lncRNA/mRNA classification, transcript sequences from GENCODE v27 [8] were collected. Classification of cancerous /non-cancerous lncRNAs has experimented on the Lnc2cancer dataset. To avoid bias in the grouping, in all datasets, the size of positive and negative samples (lncRNA and mRNAs or disease-related and non-related lncRNAs) are kept equal. A restriction of sequences of length between 3000 and 5000 nucleotides is put to alleviate the problems with unfair distribution of sequence length. The description of datasets used is presented in Table 1.

4. RESULTS AND DISCUSSION

4.1 Experimental Setup

The experiments on identification and analysis of lncRNAs are conducted with three sequence features (k-mer combination, GC content, and molecular weight) and three structure features (SSComponent, Frag_seq and MFE). The work tested 14 different combinations of these two categories of features as listed in Table 2. Other combinations produce insignificant results when applied in classification tasks. So we ignore them from the analysis.

For topic modeling, gensim [27] LDA package is used, with alpha = 0.1. Since, we assume that each document (transcript sequence) contain only one topic, a low value of alpha
Figure 3: Comparison of GC Content, molecular weight and MFE of cancerous and non-cancerous lncRNAs

Table 2: Description of the feature combinations

<table>
<thead>
<tr>
<th>Label</th>
<th>Feature Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>k-mer combination</td>
</tr>
<tr>
<td>F2</td>
<td>k-mer+GCC+mw</td>
</tr>
<tr>
<td>F3</td>
<td>k-mer+GCC+mw+SSComponent</td>
</tr>
<tr>
<td>F4</td>
<td>k-mer+GCC+mw+ Frag_seq</td>
</tr>
<tr>
<td>F5</td>
<td>k-mer+SSComponent+ Frag_seq</td>
</tr>
<tr>
<td>F6</td>
<td>GCC+mw+SSComponent+ Frag_seq</td>
</tr>
<tr>
<td>F7</td>
<td>SSComponent+ Frag_seq</td>
</tr>
<tr>
<td>F8</td>
<td>k-mer+MFE</td>
</tr>
<tr>
<td>F9</td>
<td>k-mer+GCC+mw+MFE</td>
</tr>
<tr>
<td>F10</td>
<td>k-mer+GCC+mw+SSComponent+MFE</td>
</tr>
<tr>
<td>F11</td>
<td>k-mer+SSComponent+ Frag_seq+MFE</td>
</tr>
<tr>
<td>F12</td>
<td>SSComponent+ Frag_seq+MFE</td>
</tr>
<tr>
<td>F13</td>
<td>k-mer+GCC+mw+SSComponent+Frag_seq+MFE</td>
</tr>
<tr>
<td>F14</td>
<td>k-mer+GCC+mw+SSComponent+Frag_seq+MFE</td>
</tr>
</tbody>
</table>

is preferred [1]. All other parameters are set to gensim default values. For other machine learning algorithms, Python Scikit-learn [24] implementations, with default parameters are used. The performance of classifiers is measured by accuracy, precision, recall and f-score on a 5-fold cross validation.

4.2 Significance of proposed secondary structure representations

One of the goals of current work is the identification of meaningful representations of secondary structure features. The previous work [20] used a simple binary representation proposed in [18]. Since the method does not consider the presence of various structure elements and pairing nucleotides, it fails to capture the complete semantics of secondary structure. The current work proposes two secondary structure feature representations - structure status component (SSComponent) and structural fragment sequence (refer section 3.2 for more details). In order to understand the significance of these representations, the performance of classification algorithms with these features were compared. The results are shown in Figure 1. It is clear from the figure that the proposed representations are equally good in classifying lncRNAs from mRNAs and outperform the binary representation with some exceptions.

4.3 Classification of lncRNA-mRNA using hybrid feature set

The important contribution of the current work is identification of a hybrid sequence-structure feature set for lncRNA analysis. The comparison of lncRNA-mRNA classification results are represented in Figure 2. It can be noted from the results that the accuracy is highest in case of F15 (all proposed features are used) and lowest when F1 (only k-mers) is used. Among the sequence features, GC content have important contribution than k-mer combinations whereas among the structure features, SSComponent slightly outperform other representations. It can also observe that the performance is low when only sequence features or only structural features are used (F3 and F14). The performance of all algorithms with structure features are better compared to the cases where only sequence features are used. Th best classification accuracy score obtained is around 0.78 obtained for LDA with F14. Almost close results are obtained for RF and LDA for F10-F13 (combined sequence and structure features) and an average of 0.7 accuracy and f-score is obtained for hybrid features. Though comparison of performance of machine learning algorithms is not the focus of this study, LDA and Random Forest performed better with majority of the feature combinations. Fine tuning of the algorithm related parameters may improve the results, which is not under the scope of the present study.

4.4 Classification of cancerous/non-cancerous lncRNAs

Another important objective of the present study is: can the same feature combinations used for lncRNA-mRNA classification helps to characterize cancer related lncRNAs? This analysis pay attention on the functional classification of lncRNAs. To understand the nature of cancerous and non-cancerous lncRNAs, the distribution of data is visualized (Figure 4). The visualization shows that the classes of cancerous and non-cancerous lncRNAs are linearly separable and hence the feature characterization is meaningful.
Further, we compared the GC content, Molecular weight and MFE of two classes. One can observe from Figure 3 that amount of molecular weight and MFE, have significant differences in cancerous and non-cancerous lncRNAs. High value of MFE in cancerous lncRNA indicate that their secondary structure is less stable compared to that of non-cancerous. When the length of four structure elements (stem, multi loop, interior loop and hairpin loop) in both classes are compared, average stem length in cancer related lncRNAs is less whereas the various loop lengths are high compared to non-cancerous lncRNAs. Figure 5 shows this comparison. This observation is logically supported by the analysis in [2], that loops in lncRNAs enable their epigenetic roles such as chromatin modification which leads to cancer development. This give more evidence to the studies of lncRNAs based cancer biomarker identification. The accuracy and f-score of cancer/non-cancer lncRNA classification, with the proposed feature combinations gave similar results of lncRNA-mRNA classification. The presence of hybrid feature set produce better result in all algorithms, as depicted in Figure 6.

4.5 Analysis of k-mer distribution

The current work also seeks whether the k-mer distribution follow any pattern in lncRNAs and mRNAs? When the positions of top 15 k-mers (for 4 ≤ k ≤ 7) obtained from tf-idf ranking were plotted, in all lncRNA sequences, the appearance of k-mers are dense at the beginning of the sequence and sparse towards the end. But, in mRNAs, the distribution of the same k-mer is almost dense in all part of the sequence. This distribution is observed for most of the k-mers. This behaviour implies that conserved patterns in lncRNAs are at the beginning of the sequence and their frequency is low towards the end of sequence whereas mRNAs show conservation of patterns throughout the sequence. Sample results are shown in Figures 7 and 8.
Figure 6: Cancerous/non-cancerous lncRNA classification: comparison of accuracy and f-score of four machine learning algorithms for different feature combinations in cancer dataset.

(a) 4-mer:cugg
(b) 4-mer:cagc
(c) 4-mer:gcag
(d) 5-mer:cccag
(e) 5-mer:cugga
(f) 5-mer:ggaga

Figure 7: Distribution of top $k$-mers in lncRNA and mRNA sequences. (a)-(c) are 4-mers and (d)-(f) are 5-mers.

5. CONCLUSIONS

In this paper, we presented the impact of hybrid sequence-structure features in lncRNA identification and an analysis of their functional role in cancer association. The proposed features include sequence features such as $k$-mer combination, GC content and molecular weight and structure features such as structure status component, fragment sequence distribution and minimum field energy. $k$-mer combination, structure components and fragment sequences are novel contributions of the present work, by extending concepts from literature. The performance of the different feature combinations are compared by applying in two classification tasks—lncRNA/mRNA classification and cancerous/non-cancerous lncRNA classification. This study shows that the presence of structural features improve the results and a combina-
tion of all proposed features produce the best effect. The significance of secondary structure feature representations described in this work was analysed by comparing the classification accuracy, and results prove considerable improvements. Another significant finding of the study is the functional classification of lncRNAs based on their role in cancer. The results show that the classes of cancerous and non-cancerous lncRNAs show significant differences in their structure properties (such as length of loops and stems). The results can further be extended to the functional annotation of cancer-related lncRNAs. The distribution of k-mers in lncRNAs and mRNAs put lights on to the sequence level positional conservation of patterns.

6. ACKNOWLEDGMENTS

This publication is an outcome of the R&D work under the Visvesvaraya PhD Scheme of Ministry of Electronics & Information Technology, Government of India, being implemented by Digital India Corporation.

7. REFERENCES


### ABOUT THE AUTHORS:

<table>
<thead>
<tr>
<th>Manu Madhavan</th>
<th>Gopakumar Gopalakrishnan Nair</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.jpg" alt="Manu Madhavan" /></td>
<td><img src="image2.jpg" alt="Gopakumar Gopalakrishnan Nair" /></td>
</tr>
</tbody>
</table>

Manu Madhavan is a research scholar of Department of Computer Science and Engineering, National Institute of Technology Calicut (NITC), Kerala, India. His research focus is on Long Non-coding RNA functional analysis and annotation. He received his M.Tech in Computational Linguistics from Calicut University. His areas of interests include Bioinformatics, Data-mining, Machine Learning and Natural Language Processing.

Gopakumar Gopalakrishnan Nair received his PhD in Bioinformatics from the University of Kerala, India in 2013. Presently he is working as an Assistant Professor in the Department of Computer Science and Engineering, National Institute of Technology Calicut, India since July 2010. His research interests include RNA Bioinformatics, Biological Network Analysis and Data Mining. He is a member of the IEEE as well as ACM.