Quantitative Properties of Software Systems: Specification, Verification, and Synthesis

Srđan Krstić
DEEPSE group - DEIB - Politecnico di Milano, Milano, Italy
via Golgi 42 - 20133, Milano, Italy
srdan.krstic@polimi.it
http://home.deib.polimi.it/krstic/

ABSTRACT

Functional and non-functional requirements are becoming more and more complex, introducing ambiguities in the natural language specifications. A very broad class of such requirements are the ones that define quantitative properties of software systems. Properties of this kind are of key relevance to express quality of service. For example, they are used to specify bounds on the timing information between specific events, or on their number of occurrences. Sometimes, they are also used to express higher level properties such as aggregate values over the multiplicity of certain events in a specific time window. These are practical specification patterns that can be frequently found in system documentation. The goal of this thesis is to develop an approach for specifying and verifying quantitative properties of complex software systems that execute in a changing environment. In addition, it will also explore synthesis techniques that can be applied to infer such type of properties from execution traces.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification

General Terms

Theory, Language

Keywords

Temporal logic, trace checking, verification, quantitative properties, synthesis, specifications, aggregate operators

1. INTRODUCTION

Today, software is a key element in solutions of very hard problems and as a result it is becoming very complex and hard to verify and maintain. The desirable properties of systems (and the assumed properties of the environment) are becoming more complex, as well.

ICSE '14, May 31 - June 7, 2014, Hyderabad, India

In the last decade, the research efforts in the area of software verification have focused on verifying qualitative properties of systems (e.g., safety or liveness properties). However, many important software characteristics can be quantitative, such as those related to non-functional requirements like response-time, throughput or availability. Because of this, better techniques are needed to assist in the design and implementation of reliable and correct software.

For example, model checking is a verification technique that consists of an exhaustive exploration of the state-space of a model to provide a proof that a system conforms to its specification; otherwise, it provides a counterexample in a form of a violating run of the system. Quantitative verification [18] is still an immature research area: its goal is to offer the full benefits of model checking in addition to performing quantitative evaluation to establish properties such as:

P1: *"A client is allowed to submit no more than 3 service requests each hour."* (throughput)

P2: *"The system must not have more than 2 failures per month."* (reliability)

P3: *"The average response time of a service must not exceed 30 milliseconds, if invoked by a premium customer."* (response time)

P4: *"Never allocate more then 3 machines within 2 minute time window."* (resource thrashing)

P5: *"The probability of authentication service failure is less than 0.001."* (reliability)

In my thesis I will deal with quantitative properties from three key perspectives: specification, verification and synthesis.

In the first part, I will provide a formal language for specifying this kind of properties and provide an overview of related specification patterns occurring in practice.

The first three properties above have four common characteristics: 1) they express a numerical bound on a certain value (e.g., "response time must not exceed 30 milliseconds"); 2) they consider a time-bounded sequence of past events (e.g., "since the last downtime"); 3) they refer to specific event(s) (e.g., "service requests") and/or timing relations between specific events (e.g., "response time of a service"); and 4) they (possibly) apply aggregate transformations on the multiplicity of events and/or on their timing information (e.g., "average response time"). Property P4 refers to quantitative bound on the behavior of cloud-based, elastic systems. The aspects of interest in this area are high-level

properties, such as *elasticity*, *plasticity*, and *resonance* [15]. *Elasticity* is defined in [16] as the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in autonomous manner, such that in each point of time available resources match the current demand as closely as possible. *Plasticity* and *resonance* are unwanted properties of elastic systems; the former states that a system is unable to recover its initial configuration after scaling up, while the latter refers to a situation when the number of allocated resources is continuously increasing, even when subjected to a bounded load [3]. In the last property, probability is used to quantify unreliable or unpredictable behavior of systems, for example reliability requirement of an external service in a service composition.

After identifying common set of recurring specification patterns and defining a language that can express them, I will focus on verification procedures that can be applied to it. Since non-functional requirements highly depend on the behavior of the environment, the approach mainly aims at performing verification at run time, where the real behaviors can be observed. For example, given that our system uses an external service, at design time we can only make an assumption (usually based on the service level agreement) on the response time of that service. However, during execution, this assumption can be invalidated, since response time may vary. Therefore, the approach needs to be efficient to avoid introducing a high overhead in the performance of the system. Additionally, the approach has to be general, i.e., applicable in different domains and contexts (e.g., Web service compositions, pervasive systems, etc.).

It is known that formal specification and runtime verification are not widely adopted by practitioners and most of the companies still use specifications written in natural language. Therefore, another goal of my thesis is to promote the use of formal specifications and their adoption in practice. Specification inference [23] is a process of constructing a statement offered as an explanation or justification of program behavior. Program synthesis [23] is the set of search-based techniques that generate a program that matches a given specification. Synthesis techniques will be applied in the context of the developed language to help requirements engineers automatically formalize specifications of existing systems.

2. PROPOSED APPROACH

The main contribution of this thesis is a formal methodology and the tools for the specification and verification of quantitative properties of complex software systems, as well as to promote the use of formal specifications in practice by providing specification-aid tools based on synthesis techniques. The assumptions are that applications execute in a changing environment, thus design-time verification cannot model and simulate the behavior of the underlying execution infrastructure and of the external components used by the application, which play an important role in applications' ability to satisfy its non-functional requirements. The contributions of this thesis are threefold:

1. A high-level specification language for expressing quantitative properties of software systems.

2. A lightweight runtime verification technique able to detect (or predict) violations of such properties.

3. A specification aid tool that synthesizes formal specifications from the execution traces of the existing systems.

In the following subsections, I will detail on each contribution point.

2.1 Specification of Quantitative properties

To specify in a precise way, the quantitative properties described in Section 1 it is necessary to develop a specification language with appropriate semantics.

Current state of the art languages, as detailed in Section 3, cannot completely capture these properties. My starting point is a language called SOLOIST [9], [10], developed in our research group to specify service composition interactions. It has useful constructs to express quantitative properties. For example, using SOLOIST we can write property P1 as:

$$\mathsf{G}(\mathfrak{C}^{3600}_{\leq 3}(\mathsf{req}))$$

where proposition req denotes the client request event; $\mathsf{G}(\cdot)$ is the *globally* temporal operator and $\mathfrak{C}^{3600}_{\leq 3}(\cdot)$ is a *counting* modality. It expresses the bound (≤ 3) on the number of occurrences of the req proposition within a time window (3600 seconds).

However, SOLOIST cannot express all the properties mentioned in Section 1. My plan is to extend SOLOIST by incrementally adding new modalities that will express different types of quantitative properties and extending timing features of the language to add more flexibility for the users.

The main requirements of the new language are: 1) to refer to certain events in a bounded window of time, that can be defined using constraints on timing information - time-based (e.g., "in the last 10 days") or with respect to certain events - interval-based (e.g., "since the last authentication failure"); 2) to capture timing information between specific pairs of events; 3) to perform aggregation operations on the event multiplicity (e.g., "maximum number of service invocations") and/or timing information (e.g., "weighted sum of response times"); 4) to express a bound on the result of the aggregation; and finally, 5) to express high level properties of systems, like *elasticity*, explained in Section 1.

To formalize these properties I plan to make an extensive literature overview (consisting of both scientific papers and industrial specifications) in the field of service-based applications, pervasive and cloud-based systems, to complement the study that lead to development of SOLOIST and to collect information on all additional parameters considered when expressing each of the properties. For example, elasticity parameters that need to be investigated are *speed* and *precision* [16] of the scaling process.

The final specification language will be developed to conform with recommended criteria for formal specifications [19]. In short, decidability and the complexity of the language needs to be studied to enable practical automatic verification; each decision about its semantics needs to be documented and justified; and the language needs to be usable and communicable to reasonably well-trained people.

Additionally, I intend to classify all the properties I encounter in terms of specification patterns, as done in [9].

2.2 Runtime Verification of Quantitative properties

The aforementioned approach will develop a verification procedure for the proposed specification language. The aim of the verification is to demonstrate that systems conform to the quantitative properties stated in their requirements. The road-map for obtaining appropriate tool consists of first developing off-line trace checking procedure for the new language and then a runtime verification procedure [21] and extending them incrementally as the language is constructed.

At the time of writing, an SMT-based technique for verifying SOLOIST formulae has already been developed. The implemented tool supports offline trace checking as a part of post-mortem system analysis. The current focus is on efficient runtime verification procedure. Runtime verification can be used to detect violations of specifications and trigger automatic adaptation or notify the developers about the need for offline evolution. To create an efficient runtime verification procedure I plan to investigate different encodings of past execution traces [13] to minimize the amount of bookkeeping needed to decide whether a property is violated. Other ideas include leveraging existing cloud computing infrastructure [1] and parallelizing the verification procedure over the independent subformalae of the specification. Performing incremental verification [7] can be beneficial to determine the minimal part of the system that needs to be reverified and avoid unnecessary repetitions.

2.3 Synthesis of formal specifications

While the theoretical basis for synthesis from logical specification has been known for decades, only recently a rather broad class of synthesis systems has reached the level of practical applications [11]. In the context of my thesis I will investigate methods for synthesizing formal specifications from a given set of correct runs of a system. This can be used as a specification aid in a practical scenario where companies already have execution traces of their systems that are deemed correct. The focus will be on the inference of quantitative specifications of the systems, in contrast with inference of behavioral specifications, for which there is an ample literature. The tool will analyze multiple execution traces and either infer the complete quantitative specification or synthesize values of the parameters of a specific specification pattern.

3. RELATED WORK

There are several distinct definitions of quantitative properties and approaches to their specification and verification. Finkbeiner et al. [14] propose an extension of LTL that returns values from an execution trace. They use it to compute aggregate values and collect statistics over runtime executions. Reference [2] defines an extension of metric first-order temporal logic (MFOTL) which supports aggregation. The language can express aggregate properties over the values of the parameters of relations, while the missing requirement is to expresses aggregate properties on the multiplicity of relations in the temporal first-order structure.

The approach shown in [18] defines quantitative properties as bounds on the probability of occurrence of a certain event or bounds on the expected reward in a given state of a model of the application. In this case, models are represented as discrete (or continuous) time Markov chains with rewards and properties as PCTL formulae with rewards. Another definition of quantitative properties comes from the work in [12], which considers a notion of quantitative languages as a generalization of boolean languages. A boolean language is a set of words over some finite alphabet. We can view these languages as functions that assign a boolean value to each word, depending on whether it belongs to the language. In contrast, a quantitative language is a function that assigns a real value to each word, thus a word has a *degree* of membership in a quantitative language.

The approach presented in [1] exploits a Map-Reduce framework to validate properties of traces written in LTL. This work focuses on recasting the trace checking problem into a Map-Reduce framework, by distributing verification tasks of property subformulae over many parallel sites.

Although a lot of work has been done in the field of functional specification inference (e.g., [26], [6]), inference of non-functional aspects of systems is still an immature research field. The work of Tan et al. [24] addresses the problem of obtaining the timing constraints of external services, given a global constraint of the service composition. They build a label transition system describing states of the parametrized composite service model, where the parameters represent timing constraints on the external services. Their algorithm can synthesize the values of the parameters. However, it assumes that the composite service is already formalized, and is able to synthesize only response times. In [17] authors use counterexample-guided inductive synthesis technique to synthesize parameters of signal temporal logic formulae using simulation traces of the system.

None of the mentioned work reasons on the aggregate values that are extracted from a time-bounded sequence of events, which are frequently used in practice [9].

4. PROGRESS TO DATE

In the work done so far we explored possible mappings of SOLOIST to several classes of lower-level languages like CLTLB($\mathcal{D}$), to support automated verification. CLTLB($\mathcal{D}$) is a *Constrained* PLTLB (Propositional Linear Temporal Logic with both future and past modalities) augmented with atomic formulae over a constraint system $\mathcal{D}$. This enabled us to perform trace checking using an SMT-based decision procedure for CLTLB($\mathcal{D}$) [5]. We showed that the encoding is feasible and reported the trace checking efficiency in [8].

After applying this approach to several data sets, we noticed a common property of all the real-life traces we used: they are very sparse. We measured that the number of meaningful occurrences of events was 0.0008 times smaller than the total number of time instants, the traces span. This led us to explore a more suitable low-level logic - QF-EUFIDL (Quantifier-free Integer Difference Logic with Equality and Uninterpreted Functions) and develop a more succinct encoding of the traces. The approach reported in [4] is able to check sparse traces much more efficiently then using the previous approach. These approaches are implemented as two separate plugins in the bounded model checker ZOT[1] and validated by performing trace checking on traces from real-life data sets like [20], [25], [22], showing promising results. We have reported time and memory scalability of the approaches, and the encoding complexity (linear in case of CLTLB($\mathcal{D}$) and polynomial in case of QF-EUFIDL).

[1] `http://code.google.com/p/zot/`

Ongoing work involves developing a Map-Reduce variant of the trace checking algorithm that will leverage a cloud infrastructure to parallelize the procedure.

5. EVALUATION METHODOLOGY

Evaluation of the approach will consist of its application to realistic case studies obtained from open-source repositories or, preferably, from collaborations in industrial projects. Generality of the approach will be assessed by applying it to problems from various domains, using different technologies (e.g., Web service orchestrations, Java RMI, etc.).

The real-life requirement descriptions will be used to validate the specification language. It is important to note how many requirements are expressed and the size of the each final specification formula (in terms on total number of subformulae). Also, usability of the language will be evaluated by performing a controlled experiment and recording the experience of the practitioners.

The runtime verification will be evaluated by measuring the amount of overhead it will introduce to the normal execution of the system.

Finally, the specification synthesis will be evaluated by generating traces with a certain predefined property and applying specification inference on them. We will compare the inferred specification with the original one, and perform statistical characterization on the amount of traces that are needed to obtain a certain precision in the inferred property.

6. ACKNOWLEDGMENTS

This work is supported by the European Community under the IDEAS-ERC grant agreement no. 227977-SMScom.

7. REFERENCES

[1] B. Barre, M. Klein, M. Soucy-Boivin, P.-A. Ollivier, and S. Hallé. MapReduce for Parallel Trace Validation of LTL Properties. In *Proc. of RV 2012*, volume 7687 of *LNCS*, pages 184–198. Springer, 2013.

[2] D. Basin, F. Klaedtke, S. Marinovic, and E. Zălinescu. Monitoring of temporal first-order properties with aggregations. In *Proc. of RV*, LNCS. Springer, 2013.

[3] M. M. Bersani, D. Bianculli, S. Dustdar, A. Gambi, C. Ghezzi, and S. Krstić. Towards the formalization of properties of cloud-based elastic systems. In *Proc. of PESOS 2014, co-located with ICSE*. ACM, June 2014.

[4] M. M. Bersani, D. Bianculli, C. Ghezzi, S. Krstić, and P. San Pietro. SMT-based checking of SOLOIST over sparse traces. In *Proc. of FASE*. Springer, April 2014.

[5] M. M. Bersani, A. Frigeri, A. Morzenti, M. Pradella, M. Rossi, and P. San Pietro. Constraint LTL Satisfiability Checking without Automata. *CoRR*, abs/1205.0946, 2012.

[6] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *Proc. of the ESEC/FSE*, pages 267–277. ACM, 2011.

[7] D. Bianculli, A. Filieri, C. Ghezzi, and D. Mandrioli. Syntactic-semantic incrementality for agile verification. *Science of Computer Programming*, November 2013. In Press.

[8] D. Bianculli, C. Ghezzi, S. Krstić, and P. San Pietro. From SOLOIST to CLTLB($\mathcal{D}$): Checking quantitative properties of service-based applications. Technical Report 2013.26, Politecnico di Milano - DEIB, Oct. 2013. `https://db.tt/E1hGH5Zq`.

[9] D. Bianculli, C. Ghezzi, C. Pautasso, and P. Senti. Specification patterns from research to industry: a case study in service-based applications. In *Proc. of ICSE 2012*, pages 968–976. IEEE Press, 2012.

[10] D. Bianculli, C. Ghezzi, and P. San Pietro. The tale of SOLOIST: a specification language for service compositions interactions. In *Proc. of FACS'12*, volume 7684 of *LNCS*, pages 55–72. Springer, 2013.

[11] R. Bodik and B. Jobstmann. Algorithmic program synthesis: introduction. *International Journal on STTT*, 15(5-6):397–411, 2013.

[12] K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. *ACM Trans. Comput. Logic*, 11(4):23:1–23:38, 2010.

[13] J. Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.*, 20(2):149–186, June 1995.

[14] B. Finkbeiner, S. Sankaranarayanan, and H. Sipma. Collecting statistics over runtime executions. *Formal Methods in System Design*, 27:253–274, 2005.

[15] A. Gambi, A. Filieri, and S. Dustdar. Iterative test suites refinement for elastic computing systems. In *Proc. of ESEC/FSE*, pages 635–638. ACM, 2013.

[16] N. R. Herbst, S. Kounev, and R. Reussner. Elasticity in Cloud Computing: What it is, and What it is Not. In *Proc. of ICAC 2013*. USENIX, June 2013.

[17] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. In *Proc. of the 16th HSCC*, pages 43–52. ACM, 2013.

[18] M. Kwiatkowska. Advances in quantitative verification for ubiquitous computing. In *ICTAC 2013*, volume 8049 of *LNCS*, pages 42–58. Springer, 2013.

[19] A. v. Lamsweerde. Formal specification: a roadmap. In *Proc. of the Conference on The Future of Soft. Eng.*, ICSE, pages 147–159. ACM, 2000.

[20] P. Leitner, W. Hummer, and S. Dustdar. A Monitoring Data Set for Evaluating QoS-Aware Service-Based Systems. In *Proc. of PESOS 2012*, pages 67–68, 2012.

[21] M. Leucker and C. Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293 – 303, 2009.

[22] A. Metzger, R. Franklin, and Y. Engel. Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In *SRII Global Conference*, pages 313–322, 2012.

[23] S. Srivastava. *Satisfiability-based program reasoning and program synthesis*. PhD thesis, University of Maryland, College Park, 2010.

[24] T. H. Tan, E. André, J. Sun, Y. Liu, J. S. Dong, and M. Chen. Dynamic synthesis of local time requirement for service composition. In *Proc. of ICSE 2013*, pages 542–551. IEEE Press, 2013.

[25] B. van Dongen. Bpi challenge, 2012.

[26] Y. Wang, Z. Zhang, D. D. Yao, B. Qu, and L. Guo. Inferring protocol state machine from network traces: A probabilistic approach. In *Proc. of ACNS*, pages 1–18. Springer, 2011.