ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/335984310

Property-Driven Software Analysis: (Extended Abstract)

Chapter · September 2019

DOI: 10.1007/978-3-030-30942-8_44

CITATION
1

READS
84

4 authors, including:

David Déharbe
ClearSy System Engineering
114 PUBLICATIONS **940** CITATIONS
SEE PROFILE

Julien Molinero Perez
ClearSy System Engineering
4 PUBLICATIONS **33** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Machine assisted verification for proof obligations stemming from formal methods. View project

Project Formal system modelling (railway industry) View project

All content following this page was uploaded by David Déharbe on 29 December 2020.

The user has requested enhancement of the downloaded file.

Property-Driven Software Analysis

(Extended abstract)

Mathieu Comptier David Déharbe Paulin Fournier
Julien Molinero-Perez

CLEARSY Systems Engineering, France

Keywords
Formal methods; Formal proof; Software specification.

Context. Software in industrial products, such as in the railway industry, constantly evolves to meet new or changing requirements. For projects with a lifetime spanning decades (such as the control software for energy plants, for railway lines, etc.), keeping track of the original design rationale through time is a significant challenge. The software provider may eventually lose some technical control over its product, which may then become unnecessarily complex. This may hinder beneficial architectural changes, result in the integration of over-protective measures and lead to increased costs. For safety critical systems, the risk is that the software eventually degrades up to a point where safety is no longer guaranteed in some corner cases.

Typically, for any requirement that is introduced or modified for a new version of the software, the development follows all the activities in the V cycle, from high-level requirements up to integration testing. Figure 1 zooms in on the initial design activities of the V-cycle. These are precisely the activities that are verified by property-driven software analysis: a *systematic* and *complete* analysis approach to prove mathematically that a software implementation conforms to a high-level (safety or functional) requirement. This activity detects errors early in the V-cycle, before any testing needs to be conducted, resulting in increased confidence in the design and offering the possibility to revert the loss of technical control.

Technical insights. A property-driven software analysis establishes a direct and formal link between software source and the properties expected of the system integrating the software. Any discrepancy introduced in the design phase is detected, be it in the high-level algorithmic specification of the solution, or in its implementation. Technically, the approach first identifies, in collaboration with the customer, the *key properties* that must be preserved. These are system *invariant* properties relating physical objects and logical (software) variables. The analysis then proceeds and naturally uncovers a modular and layered vision of the software and of the represented entities. Each variable contributing to the implementation of the requirement is identified, its role is expressed by *connecting properties* relating it to the concrete environment elements interacting with the software. The demonstration then consists in showing that a) the key properties derive from the set of connecting properties, b) every action of the software preserves the connecting properties, and c) every evolution of the physical elements preserves the connecting properties.

The analysis team requires as inputs the high-level requirements, usually safety properties stemming from the analysis of the undesirable events and a faithful representation of the code implementing the feature (the

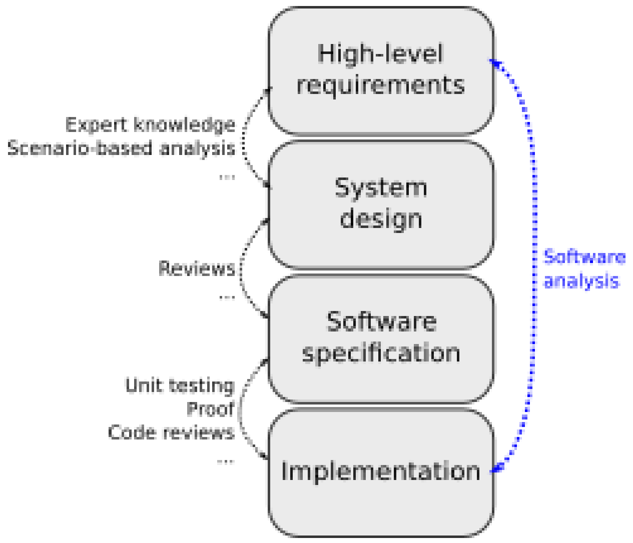

Figure 1: Position of the activity in the design process.

software specification, or a formal model of the code, if it exists, or the code itself). It is also desirable that a product expert is available part-time for one-off discussions.

Although this approach can be conducted with pen and paper with good results, we find it yields even greater benefits when the safety demonstration is carried out with a tool able to support both formal modeling of the system and mechanical verification of the demonstration. Indeed, using tools

1. guarantees that the safety demonstration is free from logical flaws;
2. uncovers all domain-specific hypotheses necessary to the demonstration;
3. requires expressing properties at the right level of detail and precision.

To formalize the system model and the safety demonstration, we have used different tools in different projects: Event-B [1] support in Atelier B [2], as well as HLL [4]. We also found it beneficial to employ Pro-B [3] animation and verification features to tune the system model. Nevertheless, the presented method does not depend on this specific tooling. All that is needed is a formalism that is expressive enough, with support for machine-controlled reasoning (i.e., interactive and automatic theorem proving).

The results of the analysis are eventually delivered as a series of documents describing the scope of the study, listing all the hypotheses made upon the environment of the studied feature, the key properties and the connection properties, a natural language version of the demonstration that the code is safe with respect to the key property, and, possibly, description of scenarios leading to unsafe situations. When formal models and proofs have been developed, these are also part of the deliverables.

Example. We illustrate this approach using the safety critical software responsible for calculating the position of a train on a track. The outputs of the software are two ordered positions p_{min} and p_{max} on the track; the key safety property is: *the physical train is totally included in the portion bounded by* p_{min} *and* p_{max}. Assume that this software computes the maximal and minimal positions (p_{max} and p_{min}) as follows:

- $p_{max} = p_{beacon} + V_{max} \cdot T$ where p_{beacon} is the position with respect to the last detected beacon, V_{max} is the maximal speed of the train, and T is the time since the beacon was detected.
- $p_{min} = p_{beacon} - R - Lg - err$ where R is the maximal movement backwards of a train, Lg is the train length and err the localization error.

Note that the result depends on the position of the last activated beacon. To ensure the key property we thus have to ensure that this position is correct (up to the localization error). Moreover, all possible train movements must be taken into account. We derive the following connecting properties:

1. A train detects a beacon activation only if its head is near the position (known by the train) of the beacon;
2. The correction added to compute maximal and minimal position covers all possible movements of the train.

Any implementation that ensures these two properties will ensure that the key safety property is satisfied. Then to show that these two properties hold in this particular setting, the proof will rely on several sub-properties that are either hypotheses, or constraints, or exported constraints such as:

a) The position of the beacon known by the train includes the correction due to the distance between the position of the detector and the head of the train;

b) A beacon is active only if it is located at the right position (*i.e.* if a beacon is dragged by a train it can no longer be active);

c) Beacon can be activated at most at a distance bounded by err;

d) Trains speed is bounded by V_{max};

e) Trains cannot go backward by a distance greater than R;

f) T is an over approximation of the time elapsed since beacon activation;

g) The real length of the train is smaller than Lg.

In summary, the initial high level requirement is first refined in two properties that combine the key property with the design principles, based on beacon placement and over approximation of the train movement. Next, the analysis yields a set of simple properties sufficient to prove the initial requirement is met.

Industrial applications This method is proposed when traditional verification and validation activities are lacking, either when case-base analysis would be incomplete for systems that contain too many states, or when defects are detected too late in the development cycle. The method has been applied on a SIL4 system for the railways, on a feature corresponding to over 100 pages of software specification and a 12kloc implementation. It has also been applied to an anti-theft device in the automotive domain, and to show compliance of smart cards with the Common Criteria level 5+ specifications.

The approach is pragmatic, with a rapid delivery of concrete results. In the traditional V-cycle, such results are eventually achieved by testing scenarios *if* the corresponding scenarios were initially identified and selected. Property-oriented analysis improves qualitatively (it is complete) and quantitatively (it applies earlier).

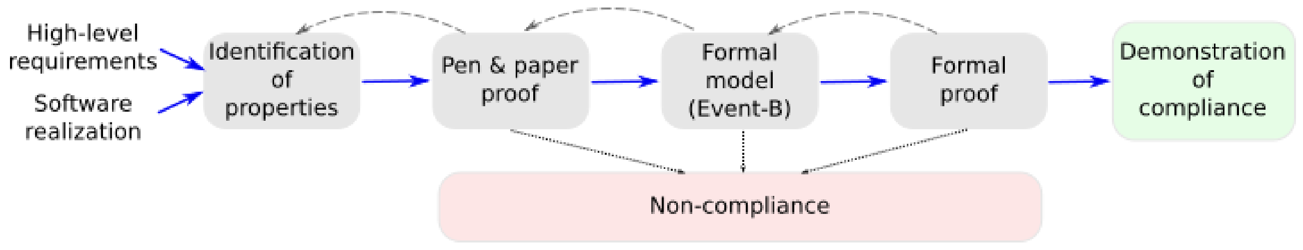

Figure 2: Synthesis of the process for property-oriented analysis

Having a property-oriented analysis of safety-critical software-based systems presents a real gain both for the provider and for the operator of that system. On the one hand, for the provider in charge of developing the system, this method provides added robustness and reduces the risk of facing problems during service. It ensures the sustainability of the knowledge of design decisions, by facilitating skill transmission. Also the safety demonstration is an important asset to obtain the certificate for the required safety level. On the other hand, the operator is also liable for the product in service and, as such, benefits from applying this approach, especially in case the product is part of a larger, multi-supplier system where it plays the role of integrator.

Synthesis We have described an application of formal methods that has seen increasing interest from different industrial partners to address the challenge of verifying systematically safety-related product features implemented in software. This rigorous approach follows the process presented in figure 2; it is driven by the identification of properties relating software entities and physical elements from the environment, which proves essential to establish key safety and functional requirements. The study of the possible changes affecting these elements provides a systematic and complete procedure to verify that these requirements are met, in all possible situations. Our experience shows that formal methods equipped with tool support, such as Event-B, can be used to support this analysis with success and provide added guarantee and effectiveness to the methodology.

References

[1] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering.* Cambridge University Press, 2010.

[2] CLEARSY Systems Engineering. Atelier B. `http://atelierb.eu`.

[3] M. Leuschel and M. Butler. ProB: A Model Checker for B. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME 2003: Formal Methods.* Springer. `https://www3.hhu.de/stups/prob/`.

[4] Julien Ordioni, Nicolas Breton, and Jean-Louis Colaço. HLL v.2.7 Modelling Language Specification. Technical Report STF-16-01805, RATP, May 2018.

View publication stats