Integrating Heterogeneous Verification of Robotics Systems

Matt Luckcuck, Marie Farrell, Rafael Cardoso

m.luckcuck@liverpool.ac.uk

The Problem

The different software components of a robotic system require different verification methods. But how do we ensure the consistency of this verification chain?

Many different formal methods are being used to verify robotic systems [1], and various software and simulation-based testing techniques are also employed. This range of approaches is needed because the different components (vision systems, agents, etc.) that make up a robotic system require different verification methods.

In Fig. 1, each component is verified using a different technique. Each verification technique makes assumptions ($\mathcal{A}$) about the component's input from the environment or other components, and guarantees ($\mathcal{G}$) properties. The dotted arrows show one component assuming the guarantee of another. This is an **implicit specification** of the system, spread over the different verification approaches.

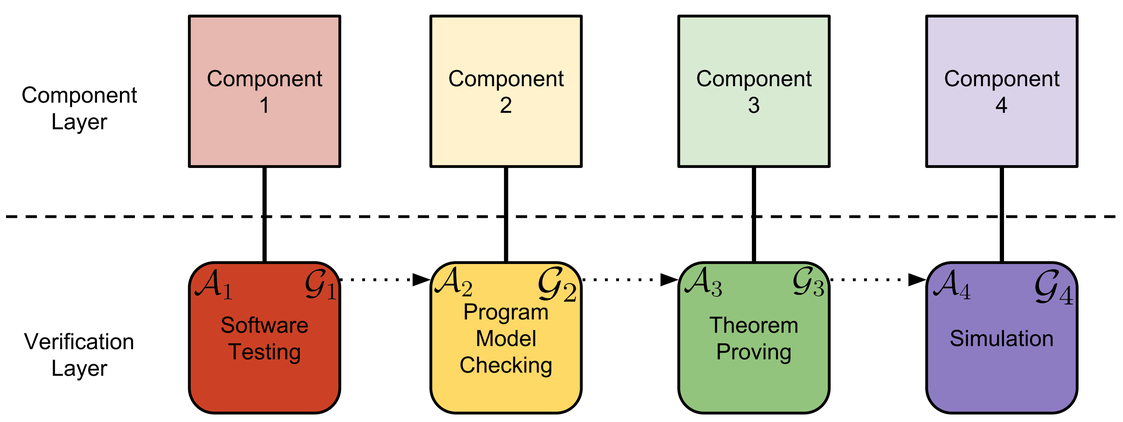

Figure 1: The implicit specification hidden in each component's verification.

How can we be sure that the guarantees are respected? Each link between the guarantee of one verification method and the assumption of another may be a translation to a new language or notation, so the link's validity depends on carefully 'importing' the meaning of another notation.

Integrating Verification Methods

Introduce an explicit system specification in First-Order Logic to describe each component's assumptions and guarantees in the same language

The Specification Layer provides an **explicit system specification**, which captures the assumptions and guarantees *hidden* in the verification of each component. This First-Order Logic (FOL) specification acts like an logical prototype for the components of the final system.

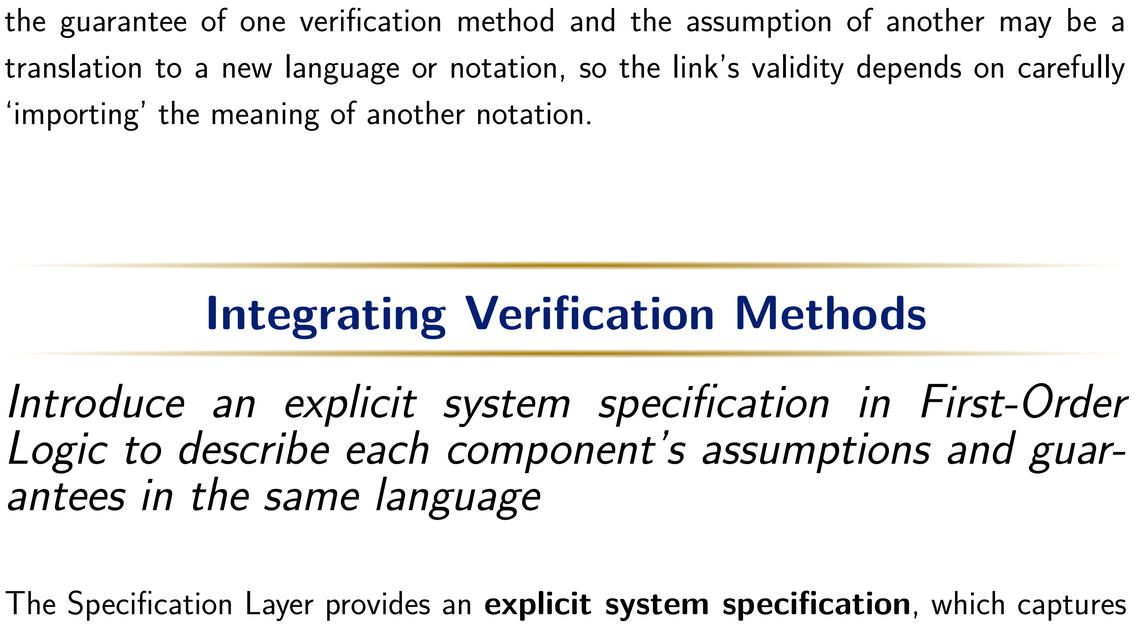

Figure 2: Introducing an explicit specification to guide specific verification.

As shown in Fig. 2, the FOL assumptions and guarantees are translated into the language or notation used to verify each different component, but the consistency of these conditions is checked first.

Benefits of this Approach

1. Top-Down: guides development from abstract specification to concrete implementation, via verification.
   Bottom-Up: check consistency of existing verification approaches.
2. Link different verification methods required by each component, and make this specification explicit.
3. Allows choice of the best verification method for each component.
4. Supports systematic, verification-led development.

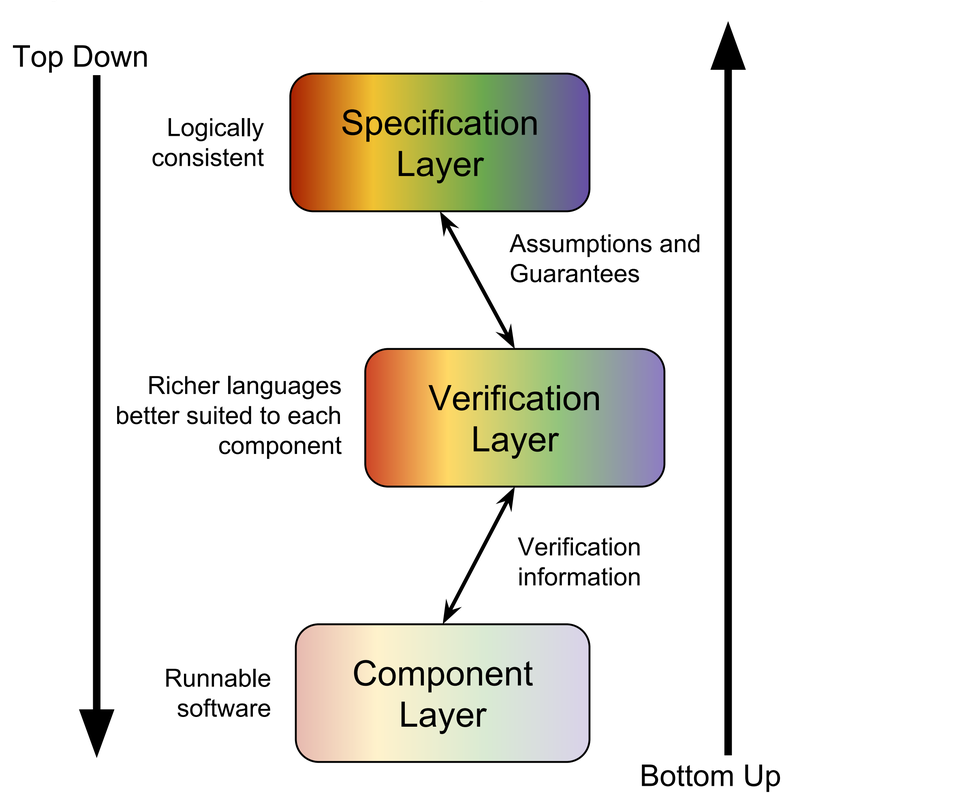

Figure 3: The benefit of each layer of the system and the information each layer provides to the others.

Various component-based frameworks support building robotic systems (e.g ROS [3] or G^{en}oM [2]) which share common abstract concepts [4]. Our approach targets components (or nodes) that can communicate, so it is applicable to many such frameworks.

Who Benefits?

Developers: get a logical prototype that provides early clarity on what each component must do.

Regulators: get a single artefact that specifies the system's behaviour, and the traceability of this specification through to the code.

Maintainers: get an unambiguous artefact that can be used to check that an update wont break the system.

Further Work

Verification Confidence: different verification methods entail different levels of confidence. How do we compare these confidence levels, and how can we decide a confidence level of the whole system?

Runtime Monitoring: monitoring that the guarantees are met at runtime could help mitigate low-confidence verification or aid fault finding.

Scale and Applicability: ensure that the approach scales, and is applicable to, more realistic systems.

Usability: ensure the approach is usable by robotic systems engineers.

References

[1] M. Farrell, M. Luckcuck, and M. Fisher.
Robotics and Integrated Formal Methods: Necessity meets Opportunity.
In *International Conference on Integrated Formal Methods*, pages 161–171. Springer, 2018.

[2] S. Fleury, M. Herrb, and R. Chatila.
GenoM: A Tool for the Specification and the Implementation of Operating Modules in a Distributed Robot Architecture.
In *International Conference on Intelligent Robots and Systems*, pages 842–849. IEEE, 1997.

[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng.
ROS: an open-source Robot Operating System.
In *Workshop on Open Source Software*. IEEE, 2009.

[4] A. Shakhimardanov, N. Hochgeschwender, and G. Kraetzschmar.
Component models in robotics software.
In *Workshop on Performance Metrics for Intelligent Systems*, pages 82–87. ACM, 2010.