A Component-based Approach to Verified Software: What, Why, How and What Next?

Kung-Kiu Lau[1*], Zheng Wang[1], Anduo Wang[2] and Ming Gu[2]

[1] School of Computer Science, The University of Manchester
Manchester M13 9PL, United Kingdom
kung-kiu,zw@cs.man.ac.uk
[2] School of Software, Tsinghua University, Beijing, China
wad04@mails.tsinghua.edu.cn, guming@mail.tsinghua.edu.cn

1 What?

Our component-based approach to verified software is a result of cross-fertilisation between verified software and component-based software development. In contrast to approaches based on compositional verification techniques, our approach is designed to solve the scale problem in verified software.

Compositional verification tends to be top-down, i.e. it partitions a system into subsystems, and proves the whole system by proving the subsystems (Fig. 1). The subsystems, often called components, are system-specific, and are therefore not intended for reuse. It follows that their proofs cannot be reused in other systems.

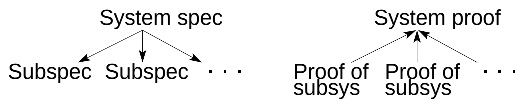


Fig. 1. Compositional verifi cation.

By contrast, our approach to verified software is bottom-up, starting with pre-existing pre-verified components, and composing them into verified composites. (Fig. 2). Components are system-independent, and are intended for reuse in many systems. Their proofs are therefore also reusable in different systems.

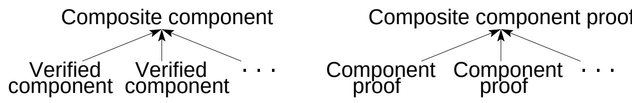


Fig. 2. Component-based approach to verifi ed software.

2 Why?

In compositional verification, the only form of ‘scaling up’ is decomposition into smaller, more manageable subsystems. The task of decomposition itself (and composing the subproofs) is directly proportional to the size of the whole system.

* Kung-Kiu Lau gratefully acknowledges fi nancial support from VSR-net (EPSRC project EP/D506735/1).

By contrast, in our component-based approach, scaling up is achieved because each step of composition is independent of the size of the whole system. The total number of composition steps required depends on the size of the whole system as well as the granularity of the components.

3 How?

The pre-requisite for a component-based approach to verified software is that components and their specification, composition and verification are not only well-defined, but also defined in such a way that verified software can be built in a component-based manner. That is, we need a component model such that it supports this approach.

3.1 A Component Model

A component model defines what components are, and how they can be composed. We have defined a component model [7] in which we can also reason about components and their composition. The defining characteristics of our components are *encapsulation* and *compositionality*, which lead to *self-similarity*. The defining characteristic of our composition operators is that they are *exogenous connectors* [8] that provide interfaces to the composites they produce.

Self-similarity is what makes our component-based approach possible. It means that our composite components have *hierarchical* specifications, *hierarchical* proof obligations, or verification conditions (VCs), and as a result, *proof reuse*, via sub-VCs, is possible.

3.2 A Case Study: The Missile Guidance System

We have implemented our component model in Spark [2], and using this implementation, we have experimented on an industrial strength case study, a Missile Guidance system [4], which we obtained from Praxis High Integrity Systems. The Missile Guidance system is the main control unit for an endo-atmospheric interceptor missile. It consists of a main control unit and input/output. An I/O handler reads data from different sensors and passes them via a bus to corresponding processing units. These units then pass their results to a navigation unit which produces the output for the system.

The implementation in [4] contains 246 packages including tools and a test harness. In total, it has 30,102 lines of Spark Ada code including comments and annotations.

Using our component model, we have implemented a component-based version of the Missile Guidance system. Its architecture is shown in Fig. 3. We reused code from [4] as computation units, and composed them using exogenous connectors. Seq1, Seq1', Seq2 and Seq4 are composite components whose interfaces are sequence connectors. Sel2 is a composite component whose interface is is a selector connector. Seq3 is a sequence connector, Sel1 a Selector, Pipe1 and Pipe2 are pipe connectors and Loop is an iterator.

We have proved the system completely, using the Spark proof tools: Examiner, Simplifier and Checker; and its proof obligation summary is shown in Fig. 4. The summary

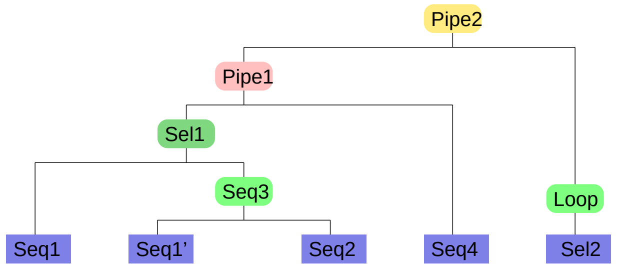
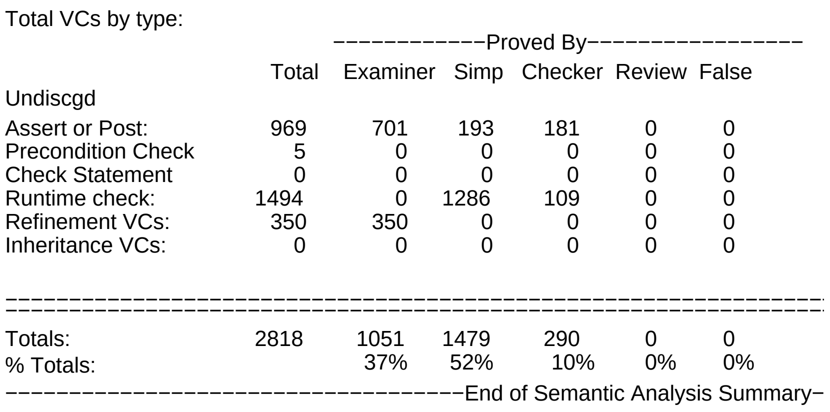


Fig. 3. A component-based missile guidance system.



```
Total VCs by type:
                               ------------Proved By-----------------
                   Total  Examiner  Simp  Checker  Review  False
Undiscgd
Assert or Post:      969       701   193      181       0      0
Precondition Check     5         0     0        0       0      0
Check Statement        0         0     0        0       0      0
Runtime check:      1494         0  1286      109       0      0
Refinement VCs:      350       350     0        0       0      0
Inheritance VCs:       0         0     0        0       0      0

=================================================================
Totals:             2818      1051  1479      290       0      0
% Totals:                      37%   52%      10%      0%     0%
------------------------------------End of Semantic Analysis Summary-
```

Fig. 4. Proof Obligation Summary of the missile guidance system.

is generated automatically by the Spark Proof Obligation Summariser (POGS). It is a summary for the VCs: their total number, types, and numbers discharged by each proof tool.

In the proofs of composite components, we succeeded in reusing proofs of sub-components, by virtue of the hierarchical nature of the VCs. We define proof reuse rate for a (composite or atomic) component simply as the ratio of the number of new VCs for the (composition or invocation) connector to the number of VCs in the sub-components (or computation unit). Of course the actual proof effort for each VC is variable, but we believe the ratio of VC numbers does give a first approximation to proof reuse rate.

As an illustration of the proof reuse rates for the component-based missile guidance system, we will show the proof reuse rates for part of the system, viz. the composite component Seq4 in Fig. 3. The subcomponents of Seq4 are shown in Fig. 5, where ‘Ibm’ is the invocation of ‘bm’ (Barometer), ‘Ias’ is the invocation of ‘as’ (Airspeed), etc.

The proof reuse rate for each sub-component of Seq4 is shown in Fig. 6. We can see that the bulk of proof efforts goes into proving the computation units of atomic components, but these proofs are only done once and can be reused afterwards. Our component-based approach is able to reuse these proofs effectively, thus reducing the cost of proof efforts of the whole system.

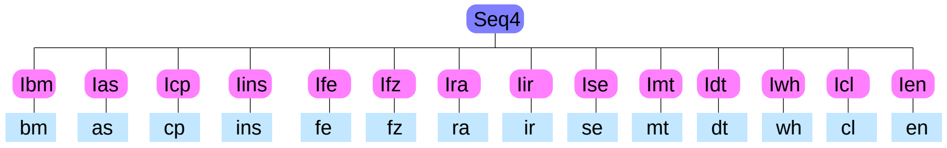
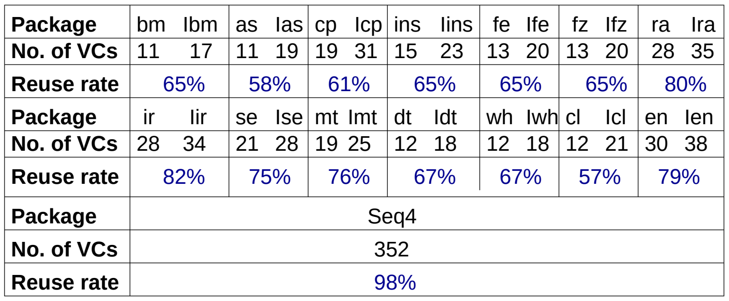


Fig. 5. Part of the missile guidance system.



Package	bm	Ibm	as	Ias	cp	Icp	ins	Iins	fe	Ife	fz	Ifz	ra	Ira
No. of VCs	11	17	11	19	19	31	15	23	13	20	13	20	28	35
Reuse rate	65%		58%		61%		65%		65%		65%		80%	
Package	ir	Iir	se	Ise	mt	Imt	dt	Idt	wh	Iwh	cl	Icl	en	Ien
No. of VCs	28	34	21	28	19	25	12	18	12	18	12	21	30	38
Reuse rate	82%		75%		76%		67%		67%		57%		79%	
Package	Seq4													
No. of VCs	352													
Reuse rate	98%													

Fig. 6. Proof reuse rates for part of the missile guidance system.

More importantly, this experiment confirms that our component-based approach can scale up, because of proof reuse.

4 What Next?

Although the missile guidance system is an industrial strength case study, our experiment is only a first step in developing and applying our component-based approach to verified software. Much more remains to be done, and here we outline some future work.

4.1 Formalisation and Proof of Properties of Component Model

A preliminary formalisation of the semantics of our component model has been done, using first-order logic [7]. To prove properties of our component model, we plan to formalise the model in a theory with a proof tool. To investigate this, we plan to use PVS [9].

4.2 Implementation in Other Languages and Tools

Implementation of our component model in other languages with proof tools, e.g. Spec# [3], JML [6], etc. will be interesting. A comparison with B [1] and its tools will also be illuminating. The objective will be to evaluate whether and how well these models and tools support our model for component-based verified software.

4.3 Larger Examples

Although the Missile Guidance System is already quite large, it is nowhere near the 1 million lines that is the target of the Grand Challenge in Verified Software [5]. Therefore, we hope to attempt increasingly larger examples, in order to produce convincing evidence that our model is fit for purpose, as far as the scale problem is concerned. By so doing we can also contribute to the repository of verified code that the grand challenge also seeks to establish.

References

1. J.R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
2. J. Barnes. *High Integrity Software: The SPARK Approach to Safety and Security*. Addison-Wesley, 2003.
3. M. Barnett, K.M. Leino, and W. Schulte. The Spec# programming system: An overview. In *Proc. Int. Workshop on Construction and Analysis of Safe, Secure, and Interoperable Smart Devices, LNCS 3362*, pages 49–69. Springer, 2004.
4. A. Hilton. *High Integrity Hardware-Software Codesign*. PhD thesis, The Open University, April 2004.
5. T. Hoare and J. Misra. Verifi ed software: theories, tools, experiments - vision of a grand challenge project. In *Proceedings of IFIP working conference on Verified Software: theories, tools, experiments*, 2005.
6. The Java Modeling Language (JML) Home Page. `http://www.cs.iastate.edu/~leavens/JML.html`.
7. K.-K. Lau, M. Ornaghi, and Z. Wang. A software component model and its preliminary formalisation. In F.S. de Boer *et al*., editor, *Proc. 4th International Symposium on Formal Methods for Components and Objects, LNCS 4111*, pages 1–21. Springer-Verlag, 2006.
8. K.-K. Lau, P. Velasco Elizondo, and Z. Wang. Exogenous connectors for software components. In G.T. Heineman *et al.*, editor, *Proc. 8th Int. Symp. on Component-based Software Engineering, LNCS 3489*, pages 90–106. Springer, 2005.
9. `http://pvs.csl.sri.com/documentation.shtml/`.