Predictable Assembly From Certifiable Components[1]

Scott A. Hissam
Software Engineering Institute

Using predictable assembly from certifiable components is one approach to developing software systems with run-time qualities that are predictable by construction. Predictable assembly combines advances in software component technology and software architecture to automate many engineering activities in constructing predictable component-based systems. In this article, I introduce the concept of predictable assembly and its connection to certifiable components, and provide a brief illustration of early experience with this approach.

Advances in technologies that support software specification and development promise dramatic improvements in the quality of software intensive systems and in the reduced cost of developing (and therefore acquiring) such systems. Progress in two broad areas is particularly noteworthy:

1. **Trusted Software Components.** It is now beyond doubt that a commercial market of software components exists, and will play an increasingly prominent role in the development of Department of Defense (DoD) systems. Recognizing this fact has led to renewed interest in the question of trustworthy components – components that are certified to exhibit known quality standards and to honor their specifications[2, 3].
2. **Analyzable Software Architecture.** While components exhibit various qualities individually, systems having such components exhibit their own emergent qualities. These emergent qualities can only be understood when a system is viewed at a level of abstraction that includes not only components but also, for example, their patterns of interaction. Software architecture technology has emerged as a way for systems designers to address the need for predictable system quality attributes at design time [1].

One theme of predictable assembly from certifiable components is to combine elements of the above two areas to provide an end-to-end method that begins with analyzable design and ends with deployed software systems that satisfy their run-time requirements. An equally important theme is using software component technology as a way of packaging and deploying the capability for predictable assembly from certifiable components into software development houses, and making these technologies easy to use by designers and developers.

This article is written from the vantage of work in predictable assembly from certifiable components (PACC) conducted at the Software Engineering Institute (SEI)[4]. Our goal is to achieve *predictability by construction*, the meaning of which is discussed later in this article. However, our work is best seen as a manifestation of – or perhaps a specialization of – a more fundamental evolution in software development practice, referred to as Model Driven Architecture (MDA) [2][5].

Both PACC and MDA are motivated by the desire to provide an end-to-end flow method from design to deployment. However, our approach to predictable assembly is to restrict designers and developers to a class of designs that are known, by construction, to be analyzable and therefore predictable. The trade off between generality and predictability must of course be made in particular development settings. Where predictability is paramount – for example in real time, secure, or highly available systems – restriction may be warranted.

Predictable Assembly

Assemblies result from composing individual software components into an interconnected collection of parts intended to carry out one or more specific functions. Often, one or more component technologies and/or protocols are used as the common unifying mechanism that permits components to be fitted together. Component technologies may be off-the-shelf such as Microsoft's Component Object Model (COM[6]), Object Management Group's Common Object Request Broker Architecture[7], Sun's Enterprise JavaBeans[8], or home grown. In any case, a common component technology is required to plug two or more components together, but it is not sufficient to ensure that the components will play well together.

To determine whether or not two components will play well together, software engineers typically look at the component's set of inputs, outputs, pre- and post-conditions, and, when available, the description of the component's assumptions about the environment (such as required processor type and speed, available memory, etc). If there is a match, the engineer will fit the two components together and hope everything works, and works well. If they do not match, the engineer may find another component and try again. To verify (or rather, gain confidence) that two components do work well together, the engineer will then test those integrated components to see if they fail. When they do fail, likely it is for reasons other than that which could have been deduced at the time the initial selection was made [3].

Predictable assembly, then, is an approach for integrating individual software components into a collection of parts where critical run-time properties (e.g., performance, safety, etc.) of that collection are reliably predicted. That is, by using predictable assembly it can be known before the actual components are integrated that they will play together with respect to one or more run-time properties of interest. This can be done if the properties of individual software components are known *a priori* to their selection or acquisition. The properties of an individual component can be the following:

- As simple as the execution latency of a function call on the component (when considering the performance of the assembly).
- As complex as a state machine description of the function call itself (when considering the safety of the assembly).

In this approach, it is the properties of individual components that are integrated together rather than the actual components. Therefore it is not necessary to actually acquire a component in advance of making the determination if it will work well together with other components. That determination is aided by a *rea-*

soning framework that is specific to a property of an assembly for which it is desired to predict.

A reasoning framework uses these properties to make a determination if the assembly of those components is well formed with respect to the rules dictated by the reasoning framework. If the assembly is well formed, then the reasoning framework generates a prediction (e.g., see the example in PACC in Action). Further, the prediction can be trusted, as the reasoning framework itself is statistically labeled to generate predictions with a stated accuracy and confidence level.

For software system developers and integrators, predictable assembly means the following:

- Reduced guesswork as to whether or not the component selection made is viable for the context in which the component will be used.
- Assemblies are predictable by construction.
- Greater confidence that components will work well together prior to testing.
- Lower likelihood that redesign, reintegration, and retesting of actual components will be necessary.

The properties that serve as input are specific to the reasoning framework. If the reasoning framework is predicting execution latencies of tasks, then the individual component latencies are required as the properties of input. If the reasoning framework is proving that an assembly is deadlock-free, then the individual component state machine might be the required property of input. From the inputs to the reasoning framework, predictions and acquisition decisions could be made. As such, it is critical that those input properties to the reasoning framework be trusted or, ideally, certified.

Certifiable Components

A component is certifiable if it has properties that can be demonstrated in an objective way. Common examples of this occur in the consumer marketplace. For example, hard disk drive (HDD) manufacturers often provide data sheets that attest to various properties of their products (e.g., seek time, average latency, or mean time between failures). Objectively, an end consumer of one of these HDDs could measure the seek time and average latency of the HDD and know whether or not its manufacturer was telling the truth. Mean time between failures would be harder for the end consumer to independently confirm, as the consumer would need all the historical data from the HDD manufacturer to reproduce the same HDD property. In this example, then, the consumer trusts that the HDD manufacturer has objectively stated these properties, and often treat them as certified properties.

Certification of a component's properties does not necessarily have to come from the component manufacturer. Consider a component that comes from the free/open source software community. An end user would be free to publish a state machine description of that component, and could even publish results that verify the component implementation matches the published state machine. This would make the state machine a certifiable property of that component. Any property of a component that can be demonstrated (in the form of a verifiable proof) or is plausible (in the form of empirical observation) can be the subject of certification.

Certification need not be a pass/fail proposition, although it is frequently treated as such. Descriptive certification of a component property (as opposed to a pass/fail normative certification) is a statement about an objective fact about a component. Revisiting the HDD example, the fact that a particular HDD has an average seek time of < 1 millisecond is not a statement that this HDD is good or bad. It is simply a stated fact, and if the consumer trusts the HDD manufacturer, the consumer can treat it as a certified property. This, then, leaves it to the integrator to determine if the value of the certified property is good enough for the assembly in which the component will be used.

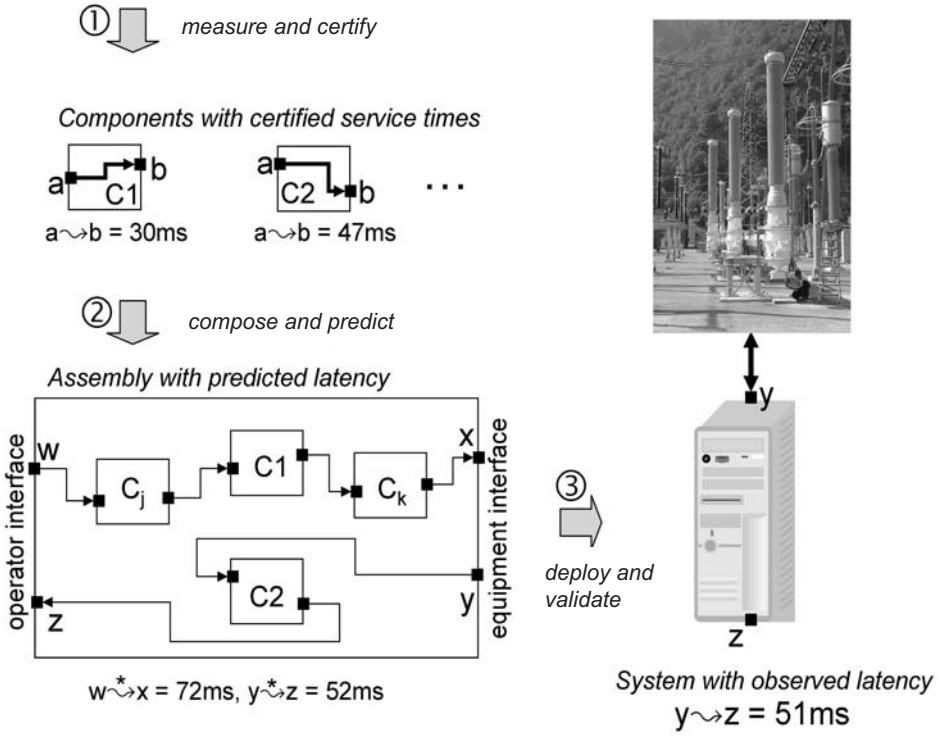
PACC in Action: A Simple Illustration

In this illustration, a software engineer wishes to predict a run-time property, execution latency, of a task with an assembly of components. The illustration is drawn from a proof of feasibility of predictable assembly for power transmission and distribution [4].

A power substation serves several purposes, among which is protection and control of primary equipment such as transformers, circuit breakers, and switches. The task for the software engineer in this illustration is to develop, from software components, a controller for a high-voltage switch. One function of the controller is to provide an interface that allows operators to manually open and close the switch. One activity in this task is to predict the time it takes for a controller to process operator requests, and the time it takes for the controller to report on a change in switch status.

The illustration in Figure 1 presents the gestalt of the software engineering task in terms of predictable assembly. Assume that a set of software components already exists, and that the service time of these components (defined as the time it takes for a component to do its work, assuming no blocking or pre-emp-

Figure 1: *A Predictable Substation Assembly*



tion) has been obtained or certified to a certain degree of trust (① in the figure). The software engineer selects a set of candidate components and composes specifications to produce a model of the controller assembly, which is analyzed and from which the execution latency of a task is predicted (② in the figure).

In the illustration, the connection from y to z is computed automatically based on the certified latency of C2 in the context of the entire assembly (w to x in this case) that may introduce blocking and preemption during run time, effecting latency. If the predicted latency satisfies requirements, the components (rather than their specifications) are composed and the resulting assembly is deployed. Predictions are just predictions because there is a possibility that they are wrong, so some validation is required of the deployed assembly (③ in the figure).

This illustration is intended to encapsulate the idea of how predictable assembly can be used in a development setting. What is not shown in Figure 1 is the level of automation supported in the assembly, prediction, and composition processes. In particular, using this example results in the following:

- Latency prediction for user-selected controller operations (e.g., from arrival of an operator request on w until the switch is signaled on x in Figure 1) is computed automatically from assembly specifications.
- The reasoning framework used to make latency predictions defines precisely what run-time properties of components must be known, and how these properties are specified and obtained. Thus, the properties of components that must be trusted are precisely those that enable predictions of assembly run-time behavior.
- The assumptions underlying the reasoning framework about how components interact with their environment and with each other are made explicit. Assemblies are well formed if they satisfy these assumptions. How well they are formed is checked automatically thus, assembly behavior is predictable by construction.
- The accuracy and reliability of reasoning framework predictions is objectively validated using statistically sound sampling and measurement. The quality of predictions is specified as a confidence interval – e.g., nine out of 10 predictions will have an upper error bound of 3 percent with 95 percent confidence.

Although this illustration is focused on execution latency, our project is concerned with more than just the timing properties of assemblies – e.g., safety and liveness (areas of current work), and reliability and security (areas for future work). Therefore, the technology our project is developing can be applied to many reasoning frameworks.

Status

The initial application of the SEI's PACC approach to predictable assembly was motivated by the challenges of using software component technology in the field of substation automation systems [4]. Although our project developed and validated a prototype infrastructure for predictable assembly, our main objective was exploratory. The primary result was an overall process model for the design, development, and validation for predictable assembly [5].

"Predictable assembly from certifiable components is not a radical concept, especially when viewed from the vantage of traditional engineering discipline."

A secondary result from this work was the development of a measurement and validation infrastructure supporting empirical validation of a reasoning framework – it is the validation of a reasoning framework that quantifies the quality of predictions produced by a reasoning framework for the user. A tertiary result from this work was the development of a prototype for predicting the latency of substation operator commands to a switch controller. This prototype ran on two platforms: a substation operator platform using Microsoft .NET, and a switch controller platform using Microsoft COM. The two platforms communicated through an industrial middleware, Object Linking and Embedding for Process Control[9], and used the International Electrotechnical Commission 61850 Standard for substation automation component type model [6].

Lessons from the initial application included the following:

- Adherence to the invariants demanded by the reasoning framework is vital.
- Development of a reasoning framework is a time-consuming proposition.

The first lesson from this list made it clear that the ability to reason (and ultimately make a prediction) about an assembly of components relies on consistency between what the reasoning framework expects to be true about the assemblies and its constituent components and the assemblies that can be created in the component technology. For example, the reasoning framework expected that components in the assemblies adhere to priority ceiling protocol [7]; however, the human designer did not always adhere to that restriction causing poor predictions. This inconsistency was spotted during validation of the reasoning framework. However, more specific rigor was clearly needed to establish and maintain consistency.

The second application of our approach (in the domain of industrial robot control, which is currently underway) is expanding, technically, to address this lesson with language (Component and Composition Language) and tool (compilers and code generators) support. The key aspect behind this additional suite of tools [8, 9] is to enforce, through automation, consistency between what is built and the invariants required by a reasoning framework.

The second lesson from this list reflects the need for expertise in the mathematical and formal models used as the foundation for any reasoning framework. As our project moves forward, it is broadening its repertoire of reasoning frameworks to include a variety of performance and verification (through model checking) technologies. Our project does this with the end goal to package these reasoning frameworks into a starter-kit to reduce the initial investment needed to create reasoning frameworks, and to make predictable assembly a practical tool for the design and deployment of software with predictable behavior.

Challenges

MDA, or something like it, is inevitable. Our project's specialized approach to MDA focuses on using software component technology to package analyzable architectural design patterns and associated reasoning (analysis) methods. As mentioned earlier, our team is developing methods and tools that will enable the software industry as well as the DoD to introduce predictable assembly from certifiable components into practice. Our team is working to demonstrate the feasibility of this approach in industrial settings, and is

seeking suitable DoD applications for trial use as well.

Although our team believes that it has demonstrated the potential of predictable assembly, there are several challenges that must be met if the ideas are to find widespread use and acceptance:

- Techniques for certifying, and labeling component properties required by reasoning frameworks must be developed.
- The business case for prediction and certification must be established, since the development of an infrastructure for predictable assembly requires upfront investment.
- The engineering methods and technology needed to build and use predictable assembly must be better understood, documented, and supported by commercial tools.

These are serious challenges, but the needs addressed by predictable assembly are real and immediate. Moreover, progress is being made, and not just at the SEI. Academic research[10] [10] and industrial practice [11, 12] are moving in the direction of predictable assembly. Further, guaranteed component quality is increasingly demanded by the marketplace, by societal needs, and by the software community's quest to establish rigorous foundations for software engineering practice.

Summary

Predictable assembly from certifiable components is not a radical concept, especially when viewed from the vantage of traditional engineering discipline. The key principle is to restrict developers to build only systems whose behaviors can be predicted, rather than trying to develop a general-purpose technology that can predict the behavior of any system. Granted, restricting developer freedom has never been an important concern of the software technology marketplace, but with the maturing of the software engineering discipline – and with the self evident importance of software to our safety and standard of living – these market forces may finally be poised to make a change for the better.◆

Acknowledgements

I would like to thank Linda Northrop, Kurt Wallnau, James Ivers, Paulo Merson, Daniel Plakosh, and Jacqueline Hissam for their helpful reviews.

References

1. Bass, L., P. Clements, and R. Kazman. Software Architecture in Practice. 2nd ed. Reading, MA: Addison-Wesley, 2003.
2. Mellor, S., and M. Balcer. Executable UML: A Foundation for Model Driven Architecture. Reading, MA: Addison-Wesley, 2002.
3. Hissam, S., and D. Carney. "Isolating Faults in Complex COTS-Based Systems." Journal of Software Maintenance: Research and Practice. John Wiley & Sons, Ltd., Mar. 1999. 183-199.
4. Hissam, S., et al. "Predictable Assembly of Substation Automation Systems: An Experiment Report." CMU/SEI-2002-TR-031. Pittsburgh, PA: Software Engineering Institute, 2002. <www.sei.cmu.edu/publications/documents/02.reports/02tr031.html>.
5. Wallnau, K. "Volume III: A Technology for Predictable Assembly From Certifiable Components." CMU/SEI-2003-TR-009. Pittsburgh, PA: Software Engineering Institute, 2003 <www.sei.cmu.edu/publications/documents/03.reports/03tr009.html>.
6. International Electrotechnical Commission. "Communications Networks and Systems in Substations." Working Draft for International Standard IEC 61850-1.10. Geneva, Switzerland: International Electrotechnical Commission, 2002.
7. Goodenough, J., and L. Sha. "The Priority Ceiling Protocol: A Method for Minimizing the Blocking of High-Priority Ada Tasks." CMU/SEI-88-SR-004. Pittsburgh, PA. Software Engineering Institute, 1988. <www.sei.cmu.edu/publications/documents/88.reports/88.sr.004. html>.
8. Wallnau, K., and J. Ivers. "Snapshot of CCL: A Language for Predictable Assembly." CMU/SEI-2003-TN-025. Pittsburgh, PA: Software Engineering Institute, 2003 <www.sei.cmu.edu/publications/documents/03.reports/03tn025.html>.
9. Hissam, S., and J. Ivers. "PECT Infrastructure: A Rough Sketch." CMU/SEI-2002-TN-033. Pittsburgh, PA: Software Engineering Institute, 2002 <www.sei.cmu.edu/publications/documents/02.reports/02tn033.html>.
10. Meyer, B. The Grand Challenge of Trusted Components. Proc. of 25th International Conference on Software Engineering, Portland, OR, May 2003. New York: IEEE Computer Press, 2003.
11. Soley, R., et al. "Model Driven Architecture." White Paper Draft 3.2. Needham, MA: Object Management Group, 27 Nov. 2000.
12. Microsoft. "Foundations of Software Engineering." Redmond, WA: Microsoft Research, 2003 <http://research.microsoft.com/fse>.

Notes

1. Sponsored by the U.S. Department of Defense.
2. See <http://niap.nist.gov/cc-scheme> for a U.S. government-sponsored effort to establish certification criteria for security-related aspects of components.
3. For Bertrand Meyer's, et al. take on the issue of trusted components, see <http://archive.eiffel.com/doc/manuals/technology/bmarticles/computer/trusted/page.html>.
4. See <www.sei.cmu.edu/pacc> for details.
5. See <www.omg.org/mda> for details.
6. See <www.microsoft.com/com> for details.
7. See <www.omg.org/gettingstarted> for details.
8. See <http://java.sun.com/products/ejb> for details.
9. See <www.opcfoundation.org/01_ about/01_whatis.asp> for details.
10. Department of Computer Science – Research. Swiss Federal Institute of Technology, Zurich, Switzerland <www.inf.ethz.ch/research/institutes/group.php?grp=Meyer#Anchor-Trusted>.

About the Author

Scott A. Hissam is a senior member of the technical staff for the Software Engineering Institute at Carnegie Mellon University. Hissam conducts research on component-based software engineering and open source software. He is also an adjunct faculty member of the University of Pittsburgh. Previously, he held positions at Lockheed Martin, Bell Atlantic, and the U.S. Department of Defense. Hissam is co-author of "Building Systems from Commercial Components" and has been published in international journals, including *IEEE Internet Computing* and *Journal of Software Maintenance*. He has a Bachelor of Science in computer science from West Virginia University.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Phone: (412) 268-6526
Fax: (412) 268-5758
E-mail: shissam@sei.cmu.edu