Is Third Party Certification Necessary?

Judith Stafford and Kurt Wallnau
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA

+1 412-268-5051 +1 412-268-3265
jas@sei.cmu.edu kcw@sei.cmu.edu

ABSTRACT

Developing software systems that are composed in total or in part from software components over which the developer has little control presents difficulties not yet addressed by component-based software engineering research. Among the problems associated with such component-based development is the potential for a component developer to misrepresent the quality of components. It is therefore paramount that some means of achieving of trust be established between component developers and component users. We are developing a model for the component marketplace that supports prediction of system properties prior to component selection. In this paper we describe the model, and describe two possible forms that the model may take in order to establish trust among participants in component-based design. We discuss the pros and cons of each choice, and leave the topic open for further discussion.

1 INTRODUCTION

Component-based development can take many forms. In this paper, we are concerned with system development that involves the use of components over which the developer has no control. Component-based design from pre-existing components presents several challenges not found in traditional software design [1]. One challenge is the need to predict the qualities of an assembly of components based on the properties of its constituent components. By "qualities" we mean a variety of quality attributes, for example performance, security, reliability. By "properties" we mean discernable features of components, for example latency, encryption, and measured test coverage. The credibility of assembly-level quality attribute prediction is a function of the credibility the quality attribute analysis technique, and the component properties that parameterize this technique. System developers must be able to trust both the analysis technique and component properties.

It is often assumed that the best way to trust in the properties of software components is through a trusted third-party certifier. Examples of this model include Underwriter Laboratories [10] and the National Infrastructure Assurance Partnership[1]. However, there are alternative approaches. Rather than vesting trust in a dedicated third party, we claim that it is possible to distribute the responsibilities of a third-party certifier among other actors in the component market, and still achieve adequate levels of trust. Different actors have roles to play in a component-based development paradigm: Component Technology Specifier, Component Technology Developer, Reasoning Framework Developer, Component Developer, and System Developer. These actors may interact in a variety of ways to achieve trust. We base this conjecture on the validity of the following analogy.

Assume that a system design must ensure that communication among users remains confidential. One way to achieve confidentiality is through a confidentiality service. Another way is to distribute responsibility for key exchange and encryption among the communicants. The former approach is analogous to using a third-party certifier, the latter to distributing certification responsibility across other participants in component-based development.

We begin by defining key terms used in this paper, and follow this with a discussion of the nature of trust and its relationship to certification. We continue with descriptions of a basic form of the model, present an extension to the basic model that supports establishing trust, and then present an alternative approach. We follow this with a discussion of some of the pros and cons of each choice and conclude by highlighting some key concerns that warrant further study.

2 TERMINOLOGY

To avoid confusion we supply definitions for key terms used in this paper before proceeding to describe the general form of the model and our experience to date in formalizing its definition.

There is no shortage of definitions for software component [1,2,3,5,9]. We adhere to the broad consensus that compo-

In Proceedings of the 4th ICSE Workshop on Component –Based Software Engineering, Toronto, Canada, May 2001.

[1] See http://niap.nist.gov/howabout.html

nents are binary implementations with an interface, but add additional criteria that reflect our concern with components over which the assembler may have little or no control.

A ***component*** is:

- an implementation of functionality that can be
- distributed in binary form and
- composed without modification according to a composition standard.

A ***component technology*** consists of a standard for developing and modeling components and a language in which to specify component assemblies.

A ***component framework*** is a conformant implementation of a component technology.

Certification is the process of verifying a property value associated with something, and providing a certificate to be used as proof of validity.

Trust is confidence in the integrity of something.

3 TRUST AND CERTIFICATION

Trust is a property of an interaction and is achieved to various degrees through a variety of mechanisms. When one purchases a light bulb it is expected that the base of the bulb will screw into the socket in such a way that it will produce the expected amount of light. The size and threading has been standardized and a consumer "trusts" that the manufacturer of any given lightbulb has checked to make certain that each bulb conforms to that standard within some acceptable tolerance of some set of property values. The interaction between the consumer and the bulb manufacturer involves an implicit trust.

In the case of the lightbulb there is little fear that significant damage would result if the bulb did not in fact exhibit the expected property values. This is not the case when purchasing a gas connector. In this case, explosion can result if the connector does not conform to the standard. Gas connectors are certified to meet a standard, and nobody with concern for safety would use a connector that did not have such a certificate attached. Certification is a mechanism by which trust is gained. Associated with certification is a higher requirement for and level of trust than can be assumed when using implicit trust mechanisms.

When we apply these notions to component-based software development, we recognize that it may also be valid to use different mechanisms to achieve trust depending upon the level of trust that is required and the cost associated with providing it.

4 THE MODEL

Trust is important when assembling components. Component users want to know that a component will function as "advertised". The user may also be interested in the long-term prospects for the component: Will bugs be resolved and patches released in a timely manner?, Will new versions be compatible with old?, etc. However, in this paper we are concerned with the question of verifying functional and quality-related values associated with a component. Therefore, we restrict our scope to the model's support for trusted interactions.

At a minimum, there are five roles required to support component-based development of systems. In the following discussion, we describe each of these roles and follow up with a discussion of the ways in which the participants interact. We then discuss alternative mechanisms for extending the basic model to address the issue of trusted component property values.

4.1 The five basic roles

Component technology specifier defines what it means to be a component as well as the types of interactions used to connect components. The resulting specification defines a standard that must be adhered to by component and infrastructure developers.

Component technology implementer provides the infrastructure that enforces the standards imposed by a component technology.

Reasoning framework developer creates analysis techniques for predicting quality attributes of component assemblies. The developer of the reasoning framework defines what needs to be known about a component in order to be amenable to the technique. It may also supply the means by which to determine the component property.

There are two primary classes of reasoning frameworks: those that depend on the execution environment, and those that can be performed given a static description of the system. Determining system latency is of the former kind and can be determined in a compositional way using rate monotonic analysis [4]. Impact analysis, on the other hand, is a static analysis technique and can be computed using compositional dependence analysis [8].

Component implementer creates components that are conformant with some component standard. Conventional component interfaces will be augmented to associate arbitrary properties and property values with components, using a mechanism such as <property,value,credibility> credentials [6].

System developer assembles components to fulfill some high-level function. The system developer expects to be able to predict the quality attributes of a proposed design before commitments are made to use specific components.

4.2 Interactions among the Roles

Figure 1 shows the five basic elements of a component technology along with their interactions. The arrows indicate the direction of the interaction; their semantics is described below.

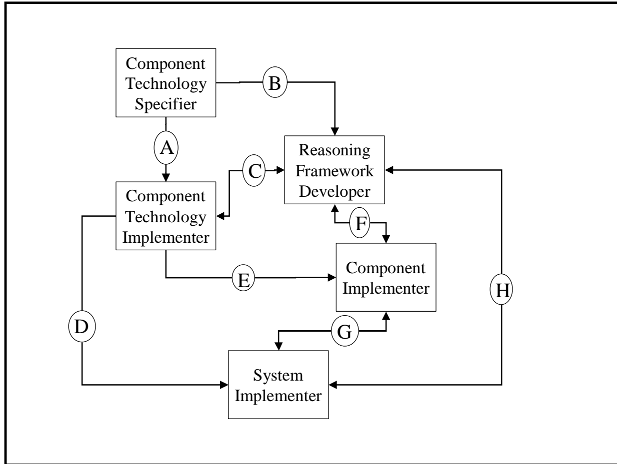

Figure 1: Five basic roles of a component-based development process and their interactions.

- Interaction A: The component technology implementer uses the specification provided by a specifier of the component technology to build conformant component infrastructure.
- Interaction B: The component technology specifier may suggest that a particular analysis technique be developed to support reasoning about some quality attribute that is important to the types of systems likely to be developed using this framework.
- Interaction C: The component technology implementer may suggest new types of analyses, and the developer of the reasoning framework will provide input as to what types of analyses are useful within certain types of systems.
- Interaction D: The component infrastructure provider must provide the system implementer with trusted infrastructure properties.
- Interaction E: The component implementer receives its standards information from the implementer of the component technology.
- Interaction F: The developer of the reasoning framework defines component properties that must be trusted, while the organization that actually build the component may be required to divulge otherwise hidden implementation details to support a particular analysis technique.
- Interaction G: The developer of the system receives components and property documentation from the component implementer, and may notify the component implementer of the desire to use an analysis technique for which a component under consideration has not been validated.
- Interaction H: A reasoning framework developer supplies the system developer with the necessary algorithms for performing the compositional analysis techniques, and the system developer may indicate to the developer of the reasoning framework that a new type of analysis would be useful.

4.3 Trust

Trust is a quality attribute we wish to affix to interactions that transpire in the component marketplace (just as confidentiality is a property we wish to affix to interactions in our earlier analogy).

A typical mechanism for developing trust between otherwise unrelated parties is a trusted third party such as Underwriter's Laboratory. In Figure 2 we show our original model extended with the addition of a Component Property Certifier.

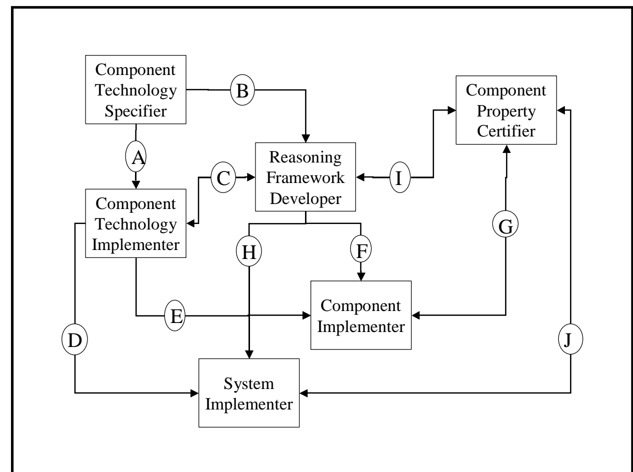

Figure 2: The model extended to support trusted interaction.

Component property certifier acts as a trusted third party. A property certifier might verify credentials that were provided by the component implementer along with components or, alternatively, create an additional credential. The component implementer trusts the certification organization with the source code for a component and the system developer trusts that the certificates supplied along with components and obtained through a certifier are valid.

The original communication between the component implementer and the system developer has been replaced by an interaction between the component implementer and the component property certifier and two new two-way interactions have been added.

- The redirection of interaction G forces the supply of components to go through the certifier. This is probably not optimal as is discussed further in the Section 5.
- The two-way interaction I between reasoning framework developer and property certifier represents the mutually informing relationship between the developer of compositional analysis techniques and the user of them. The reasoning framework developer might devise an algorithm that requires knowledge of component internals that are not certifiable. In this case the property certifier needs to notify the developer of the analysis technique so that adaptations to the algorithm can be explored.
- The two-way interaction J between the property certifier and system developer indicates that a request from the developer for components with specific certificates and property certifier's supply of those components.

This model, as depicted in Figure 2, is analogous to the "confidentiality service" available via yahoo's mail service. This service assures confidentiality among participants when communicating over the Web via a secure 3rd party.

There are other ways of achieving confidentiality over the Web. For instance, two participants can communicate confidentially through their use of private keys and encryption. So, why should we assume there is a "certifier" role in the component marketplace model that provides "trust?" Perhaps there is a duality here: we can either achieve trust in a property through a dedicated component, or through the distribution of the logical responsibilities of that component among the participants, i.e., over a pattern of interaction.

We introduce the notion of *active component dossier*, or dossier for short, in which the component implementer packs a component along with everything needed for the component to be used in an assembly. A dossier is an abstract component that defines certain credentials, and provides benchmarking mechanisms that, given a component, will fill in the values of these credentials.

A dossier is customized for particular types of components. Our desire to support customization is based on the fact that certain types of components are more likely to be used in certain types of systems for which certain types of analyses are appropriate. For instance, a particular dossier might contain an audio input component that one would expect to be used in audio applications where performance is an issue. Therefore the dossier contains, at a minimum, a postulated credential that provides an expected range for a latency measure, and a test harness that is adaptable to the environment in which the component is to be deployed so that the validity of the latency measure can be tested *in locus*.

As mentioned in Section 4 in the context of our description of the role of the reasoning framework developer, there are properties that can be determined from the static description of the system alone. For each of these properties the dossier contains a credential that includes the property name, the value, and the name of the tool used to determine the value.

What is important is that the dossier supports performance of assembly-level analysis before commitment is made to acquire a specific component. The measure of a component property must be automated, and the measures associated with a component must be reliable.

The use of the active dossier to achieve trust allows the interaction pattern shown in Figure 1 to remain. However, the responsibilities associated with the component implementer and the system developer are extended to include the packing, unpacking, and use of the dossier.

5 DISCUSSION

The notion of active dossier grew out of our experience in trying to package latency measures with components. We discovered that there are certain aspects a component's structure, such as minimal cycle time, that can affect the way in which the component's latency should be measured. While average expected latency can be estimated in the component development environment, verification of these measurements must take place in the proposed deployment environment, thus we create the dossier as a means to provide a system developer with the information and tools necessary to validate average component latency.

The dossier mechanism is more scaleable than a third party certifier because it distributes the effort and avoids a central service bottleneck. It also supports the notion of making component selection decisions that are contingent on the verification of postulated credentials within the deployment environment. However, there are unresolved issues associated with the model. For instance, it is not clear who should define the contents of a given dossier. In fact, it may well be the case that several participants will have a say and participation in the decision-making process will require extra effort by all parties.

6 OPEN QUESTIONS

There are many issues that must be addressed before the definition of active dossier is finalized. These issues include determining how to create a tamperproof dossier, how to certify measurement techniques so that the system developer can put trust in measurements made by the component implementer, and how to define an economy of

components that includes the notion of contingent purchases for those cases when a measurement must be made by the system developer.

We realize that there are many other important issues related to the idea of using an active dossier to achieve trust. We leave this as an open topic. Additionally, there are other basic questions of trust that we feel are important areas for discussion. For instance, What level of trust is required under different circumstances? Are there other mechanisms that might be used to support trust? If so, are there different levels of trust associated with them and can knowledge of these differences be used to direct usage of different mechanisms under different conditions.

7 REFERENCE

1. F. Bachman, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord and K. Wallnau. Volume II: Technical Concepts of Component-Based Software Engineering. Technical Report CMU/SEI-2000-TR-008. Software Engineering Institute, Pittsburgh, Pennsylvania, May 2000.
2. J. Cheesman and J. Daniels. *UML Components: A Simple Process for Specifying Component-Based Software*. Addison-Wesley, Boston, Massachusetts, 2000.
3. G.T. Heineman and W.T. Councill (eds.). *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley, Reading, Massachusetts, 2001.
4. M. Klein and J. Goodenough. Rate Monotonic Analysis for Real-Time Systems. Technical Report CMU/SEI-91-TR-006. Software Engineering Institute, Pittsburgh, Pennsylvania, 1991.
5. J. Rumbaugh, G. Booch and I. Jacobson. *The Unified Modeling Language Reference Manual (UML).* Addison Wesley Longman, Inc. December 1998.
6. M. Shaw. Truth vs Knowledge: The Difference Betweeen What a Component Does and What We Know It Does. *Proceedings of the 8th International Workshop on Software Specification and Design*, March 1996.
7. J.A. Stafford and K.C. Wallnau. Predictable Assembly from Certifiable Components. Technical Note CMU/SEI-2001-TN-001.
8. J.A. Stafford and A.L. Wolf. Annotating Components to Support Component-Based Static Analysis of Software Systems, *Proceedings of Grace Hopper Conference 2000* (on CD-ROM), Hyannis, Massachusetts, September, 2000. Also available in hard copy form as University of Colorado technical report CU-CS-896-99.
9. C. Szyperski. *Component Software Beyond Object-Oriented Programming*. Addison-Wesley, Boston, Massachusetts and ACM Press, 1998.
10. UL 1998, 2nd ed. 1998. *UL Standard for Safety for Software in Programmable Components*. Underwriters Laboratories, Inc. Northbrook, IL.