In Proceedings of the *4th International Conference on Electronic Commerce Research (ICECR-4)*, 505–514, Dallas, ATSMA, IFIP, 2001.

Trust-Based Procurement Support for Software Components*

Peter Herrmann
University of Dortmund
Computer Science Department
44221 Dortmund, Germany
Peter.Herrmann@cs.uni-dortmund.de

Abstract

Component-structured software facilitates the design of problem-specific software solutions for a reasonable price. Due to the significant number of principals involved in the component development and employment process, however, a new class of security problems is introduced. In particular, a malicious component is a threat to any application incorporating it. Thus, a customer of software components has to attach importance to security aspects. Unfortunately, often the available information does not suffice to perform a decent procurement decision. Therefore components have to be evaluated by means of certification and runtime monitoring. These methods, however, are usually laborious and costly. In order to reduce the expense of evaluating components, we apply an approach which takes the experience of other customers with a component in question into consideration. It employs the concept of trust management enabling to calculate trust values (i.e., values describing the trust in a component) from good or bad evaluations with it. Particularly, we introduce a trust information service collecting expertises which component customers and certification authorities gained from certification of a component as well as monitoring it during deployment. From these evaluations a trust value is generated and offered to parties interested to purchase the component. Moreover, we outline an extension of a runtime monitoring software which enables automatic generation of good or bad monitoring expertises. Likewise, the intensity of the runtime observations about a component may be adjusted due to the current trust value of the component.

Key Words: Software component, component procurement, trust management, trust information service, runtime monitoring.

*This work was funded by the German research foundation DFG

1 Introduction

In spite of its novelty, the approach of component-structured software gains more and more popularity. It facilitates the easy and cost-effective creation of applications built from independently created components (cf. e.g. [38]). Moreover, component-based systems can be tailored to the special needs of their customers and varying requirements during runtime result in dynamical changing of components and their couplings. A component-structured system is not purchased as a monolith. Instead, different developers create the components separately and offer them on an open market. The target application designer selects and buys suitable components, probably from various sources, configures them according to the particular needs of the application, and couples them to the final software product. The combination process utilizes the concept of explicit contracts. A contract is legally binding and describes agreed properties of a component and, in particular, its interface. According to [2], a contract consists of four parts specifying the structure of a component interface (i.e., methods, input and output parameters, exceptions), the desired behavior of the component and its environment, synchronization aspects, and quantitative quality-of-service properties. As another means to support system composition, reflection and introspection [37] of components is provided as well (i.e., components contain special methods enabling the exploration of component properties, methods, and interfaces at runtime). Furthermore, comfortable coupling is facilitated by scripting languages and visual application builder tools (e.g. [27]).

Meanwhile, some platforms for component-structured software are available. Best-known are Java Beans [37] and Enterprise Java Beans (EJB) [36]. Moreover, in PC-based environments COM/DCOM [29] is well-established, too. Finally, the CORBA initiative extended its platform in order to support component-structured applications [33]. All platforms provide notions for describing component types, parameter types, and interfaces. Furthermore, means to introspect components as well as coupling support are also offered.

Like other software, components are usually sold as executable code. Since their contracts as well as other documentation can be delivered electronically, too, they are perfectly suited for being traded on electronic markets. As an alternative, a component vendor may execute a component on a remote host and offer it as a telecommunication service. Here, the application owner purchases just the service in order to realize a distributed

component-structured system. A first system supporting the lookup of components and services is proposed in [13].

Component-structured software, however, imposes new security risks since, compared with ordinary object-oriented systems, it introduces new principles and roles. Besides users and application owners, also component vendors, component service providers as well as application builders are involved in the development and deployment of the software. Yet, the principals cannot trust each other to full extent since everybody may exploit the software and its components in order to get an unwarranted advantage and therefore forms a potential threat. In particular, someone may maliciously alter a component spoiling the security of the whole component-structured system. For instance, a component may be changed in order to leak confidential information to a person not permitted to read it. Thus, to reduce the risk, a component forms for an application incorporating it, the application owner procures components only if she has sufficient trust in the wellmeaningness of the component vendors. According to [39], an important condition to carry out a purchase is that "the item sold corresponds to its descriptions and is suitable for its intended purpose".

This paper is centered on an approach to support component procurement decisions by combining trust-management [18] with runtime monitoring and certification. Application owners may inform a so-called trust information service about positive and negative evaluations of component certification and runtime monitoring. From these expertises, the service calculates a so-called trust value stating the amount of trust users have in a certain component. This value is offered to potential customers who may utilize it for their procurement decisions. Moreover, the trust value can also be used to decide about the intensity of component runtime monitoring.

In the sequel, we will outline major security aspects of component structured software as well as trust management. Thereafter we sketch our trust information system in order to support procurement decisions. In the following sections, mechanisms to register components and to inquire trust values are introduced to greater detail. Finally, we outline an extension to runtime monitoring in order to automate expertise generation and monitoring control.

2 Component Security

Distributed component-structured software imposes new security aspects caused by the high number of different principals. Moreover, it includes also security problems of local applications, distributed systems, and mobile code applications. Lindqvist and Jonsson [26] developed a taxonomy of security risks for components comprising the following aspects:

- Component design: A component may contain inadvertent or intentional design flaws forming security risks for the incorporating applications (e.g., a Trojan Horse, i.e., additional hidden functions damaging the component environment). Another problem is caused by inadequate or incorrect component interface documentations preventing secure integration or deployment of components.

- Component procurement: Due to insufficient information about security aspects of components, it is difficult to decide if a component conforms with the customer's real security requirements. Furthermore, if a component is delivered via an insecure channel, it may be maliciously manipulated by a third party.

- Component integration: If a component is integrated into an application without fully understanding the preconditions for secure operation, a vulnerability may be imposed to the application. Moreover, two components may be incompatible with respect to their security levels (i.e., cooperation is only possible if essential safeguards are abandoned).

- Distribution of the component-structured system: Data between components may be wiretapped, modified, or destroyed during the transfer via an insecure network. Moreover, a network connection may be exploited for intruding a host computer in order to attack a component residing on the host.

- System use: A component-structured application or single components may be used in a way not intended by the component developers. Thus, security possibly relies on inadequate security mechanisms.

- System maintenance: Dynamic modification or extension of components may lead to side-effects affecting system security. Furthermore, like the components themselves, updates are subject to modifications during delivery, too.

A further security aspect, not listed in the taxonomy, is the mutual threat that a component is attacked by a host computer executing it and vice versa (cf. [9, 21]). Moreover, component vendors have to be protected

against wrong accusations due to spite of application administrators, other component vendors, and host operators. Finally, a component vendor has to be protected against unlicensed deployment of components.

In this publication, we address the security aspects of component procurement. The main problem here is the lack of suitable information in order to make sensible purchase decisions. At first, the customer needs information to decide if a component of interest supplies adequate security mechanisms in order to guarantee the requirements of the application owner. At second, the customer has to gain trust in the correctness of the information (i.e., the component must act in accordance with it).

Useful for component evaluation are the explicit component contracts (cf. [2, 38]) which can be enriched with formal descriptions of security relevant obligations (cf. [22, 23]). The first aspect, that a component has to be adequate with security requirements of the application owner, can be performed by proving formally that the obligation specifications and their combinations fulfill formal descriptions describing the security requirements. A first tool-supported solution for verification that the information flow in a component-structured system does not violate the requirements of the application operators, is introduced in [14]. Here, we address the second aspect guaranteeing that the customer has sufficient confidence in a component contract in order to use it for her procurement decision.

A method to gain trust in information about a software component is certification which may be performed either by the customer herself or by a certification authority. In a customer-based certification the application owner performs various checks in order to determine the suitability of the component for the application, its quality, and its impact on the system incorporating the component. In particular, one checks that the component acts in accordance with its contract descriptions. Voas [41] proposes black-box testing, software fault injection, and operational system testing as techniques for certifying component reliability and security. In black-box component testing the component is executed with various test inputs and a so-called oracle decides about failures (cf. [30]). In system fault injection data propagated between components is voluntarily corrupted in order to provide worst-case predictions in case of malicious component behavior (cf. [42]). In operational system testing the complete system is executed in a sand box (cf. [44]) as a complementary means to determine the impact of the component on the system. Other methods to check components are code inspections (white-box testing) and byte-code verification. In a code inspection the source code of a component is walked through in order to detect implementation faults. Often, however, code inspections are impossible since many components are delivered without source code. In contrast, in a byte code verification executable code is checked for security flaws. If techniques based on virtual machines are used (e.g., Java), this analysis can frequently be performed with an acceptable expenditure due to the source code-like shape of the byte code and powerful tool-support.

Alternatively, the quality checks may be performed by an authentication authority. If the tests performed by this trusted third-party were successful, it issues credentials to interested principals. If a component customer has confidence in the authority and its testing techniques, she can assume that a component provided with a credential is secure.

Another method to gain trust in components is runtime monitoring. A so-called software wrapper [10] is a piece of code extending a component. While the wrapper does not change the behavior of the component, it monitors the component interface for security flaws. We extended this concept to generic security wrappers [17]. These wrappers contain security behavior descriptions of the component contracts which are checked for compliance with the real actions at the component interface. If a wrapper detects that an action is not in accordance with a contract description, it immediately isolates the component and notifies the application operator. While the mean runtime overhead of the security wrappers is only 5%, one may reduce the intensity of the checks if the trust in a component raises due to long-lasting correct behavior.

The approach introduced below goes a step further. It facilitates to utilize experience, other component users gained from certification or runtime monitoring. It is similar to the concept of Label Bureaus [35] used to label web pages in order to protect children from unintentional access to objectionable contents. The labels for the web-pages result from self-ratings of the web page designers (first-party), community ratings by interested users (second-party), and rating by trusted authorities (third-party). With respect to security objectives, however, first-party ratings are futile since a malicious principal would never confess his real intentions. Ratings from component users can be classified as second-party while expertises by certification authorities are third-party ratings.

3 Trust Management

Khare and Rifkin [24] predict that the World Wide Web "will soon reflect the full complexity of trust relationships among people, computers, and or-

ganizations." Trust management is a new philosophy in order to determine the adequacy of trust relationships and therefore to protect the Web and other open, decentralized systems against malicious behaviors. According to Jøsang [18] trust can be gained in human interaction as well as in interaction of humans and computer systems. Humans are called *passionate entities* while things like computers, algorithms, etc. without a free will are named *rational entities*. Two separate kinds of trust are defined:

- Trust in a passionate entity: A passionate entity A trusts another passionate entity B if A believes that B behaves without malicious intent.
- Trust in a rational entity: A passionate entity A trusts a rational entity B that it will resist malicious manipulations caused by an external passionate entity C. This reflects that B cannot be honest or malicious itself since it has no free will but it can be built in a way that it acts not maliciously and withstands hostile attacks.

Moreover, Beth et al. [1] distinguish two further kinds of trust:

- *Direct trust*: An entity A trusts directly in another entity B if it believes in the capabilities of B itself.
- *Recommendation trust*: An entity A trusts in the recommendations of B with respect to a third entity C if it believes that B will give a reliable and honest assessment of the capabilities of C.

Trust in a passionate or rational entity is based on gaining many positive experience with the entity behavior while the negative experience should be low. This can be expressed mathematically by trust values expressing belief, disbelief, and uncertainty in an entity. The description of uncertainty is necessary since with only small knowledge in an entity it cannot seriously be assessed. In [19], Jøsang defines a so-called opinion-triangle (cf. Fig. 1) for modeling trust values. Belief, disbelief, and uncertainty are specified by the values b, d, resp. u which are real numbers between 0 and 1 each. Moreover, $b + d + u = 1$ holds always and the trust in an entity is stated by a point in the triangle. The perpendicular models the uncertainty in an entity while the belief or disbelief is described by the horizontal. A trust value of an entity based only on low knowledge is modeled by a point close to the top while points on the right or left bottom state great belief resp. disbelief which is based on a large number of evaluations.

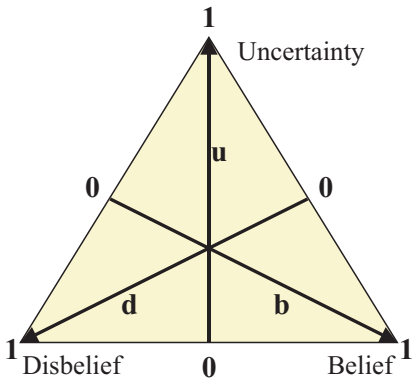

Figure 1: Opinion Triangle (taken from [19])

The trust values are calculated from the number of good or bad expertises of the entity in question. Assuming that the values p and n describe the numbers of positive resp. negative events with an entity, Jøsang and Knapskog [20] calculate the three trust values by means of the following formulas:

$$b = \frac{p}{p+n+1} \qquad d = \frac{n}{p+n+1} \qquad u = \frac{1}{p+n+1}$$

Thus, a negative event can be compensated by some positive expertises and the metric expresses a relatively liberal trust management philosophy. In contrast, Beth et al. [1] define an unforgiving policy. The probability that the belief b exceeds a certain value α is expressed by the formula

$$P(b > \alpha | p, n) = \begin{cases} 1 - \alpha^p & : \ n = 0 \\ 0 & : \ n > 0 \end{cases}$$

and the disbelief and uncertainty are calculated from b by the formulas

$$d = \begin{cases} 0 & : \ n = 0 \\ 1 & : \ n > 0 \end{cases} \qquad u = \begin{cases} 1 - b & : \ n = 0 \\ 0 & : \ n > 0 \end{cases}$$

Thus, a single negative experience destroys the trust in the entity forever. If nothing bad happens, the amount of belief depends on the number of good events. Many other metrics are sensible, too, and the rigorousness of a metric should depend on the consequences, a breach of confidence has for the entities to protect.

Trust management can be used for several application domains and in [1] key generation, authorization, keeping secrets, certification, clock synchronization, and program validation are listed as possible areas. In the field of authorization systems first solutions are implemented (e.g., [5]). Here, access to a resource does not depend on traditional access control mechanisms. Instead, a caller has to show credentials issued by third parties stating the direct trust these parties have into the caller. Depending on these values as well as on the recommendation trust in the third parties, the resource owner may decide about granting or rejecting access. Implementations on trust management-based authorization systems comprise PolicyMaker [6], REFEREE [8], and KeyNote [4]. Another realization are the Label Bureaus [35] outlined in Sec. 2. Here, the labels for evaluating the content of web pages are calculated depending on the ratings of the various parties as well as on the trust in these parties.

4 Trust Information System

In order to utilize experiences users made with a component, we need a trusted third-party collecting expertises of good and bad component behavior, calculating trust values from the expertises, and passing trust values to interested customers. The tasks are realized by the *Trust Information Service* delineated in Fig. 2. This service corresponds with component vendors, interested principals purchasing and deploying components, and authorities certifying components in behalf of a vendor or user.

Component vendors may register any software component offered commercially or free of charge declaring themselves to be in agreement that the trust information service collects expertises about the component and offers the corresponding trust values to interested parties. Of course, the decision to register a component is voluntary. Nevertheless, since registration shows confidence with the offered product, it is an effective marketing instrument. Furthermore, due to the increased information customers may decide to buy only registered and evaluated products and therefore, de facto, enforce component vendors to register.

Parties interested in information about components inquire their trust values from the trust information service. Moreover, the trust information service offers an alarm service notifying all interested component users immediately about malicious experiences gained with a component. To get a large number of expertises from varying sources, the trust information services calls all customers in intervals for reports about the component

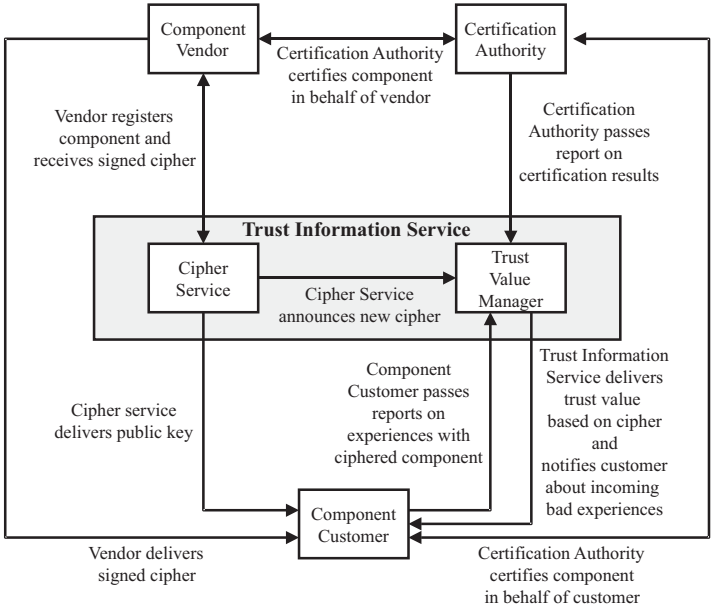

Figure 2: Trust Information Service

behavior during runtime. Furthermore, if a component user detects a serious incident indicating malicious component behavior, she informs the trust information service immediately in order to warn other users.

To guarantee a high degree of privacy for the component producers, the expertises of components are not stored together with complete component descriptors but are separated. Thus, the trust information service consists of two parts: a *Cipher Service* and a *Trust Value Manager*. If a component vendor registers a new component, the cipher service stores the relevant data and creates a unique cipher. In contrast, the trust value manager performs the storage of expertises, calculation of trust values, transmission of trust values and alarm messages, as well as the inquiry of expertises based on the ciphers. This separation of component identifiers and expertise data between two parts provides privacy since neither the cipher service nor the trust value manager have complete knowledge about the components. In contrast, the component customers and certification authorities have complete knowledge which, however, is limited to the very small amount of components employed or certified by themselves.

5 Component Registration

If a component vendor decides to subject a component to the trust-based evaluation process, he first registers it with the trust information service. The registration procedure has to ensure that the vendor does not manipulate a component between registration and delivery in order to cheat the trust information service. Moreover, one has to prevent third-parties to change components during delivery to customers or registration. These tasks are guaranteed by employing digital signatures (e.g., [34]) which are realized by the Java Cryptography Architecture [11]. A digital signature of a data set is performed in two steps: At first, the data transmitter generates a hash-value of the data set, the so-called message digest, by means of the hash function SHA-1 [32]. Thereafter, he encrypts the message digest using his private key by applying the Digital Signature Algorithm (DSA) [31] and transmits the resulting digital signature together with the data set. The receiver decrypts the signature with the public key of the transmitter. Finally, she hashes the data set herself and compares the two hash-values. If the hash-values are different, the data was altered during transport. A malicious third-party cannot create a digital signature compatible to his manipulations since he has not access to the private key of the transmitter.

In the first step of the registration procedure the component vendor creates a digital signature A of the component to register and transmits A together with the component to the cipher service which checks the hash-values. If the component was not manipulated, the cipher service creates a unique cipher of the component and stores it with other relevant data of the component and its vendor in a local database. Moreover, protected by a digital signature B, the cipher is sent to the trust value manager which adds a new entry to its trust value database. The cipher service creates a third digital signature C from the record of A and the cipher. C is transmitted to the component vendor who delivers it to persons interested in the component. Since C contains the digital signature A which was encrypted by the private key of the cipher service, the vendor cannot alter the component anymore without notifying the cipher service. The cipher service, however, does not register an altered component without creating a new cipher. Therefore a customer can compare the hash-values of signature A which is included in C and of the procured component by using the public keys of the cipher service and the vendor. If the hash-values are identical, she has confidence that the cipher in C really belongs to the procured component and that the component was not changed during delivery.

6 Trust Value Inquiry

If a customer wants to assess various components in order to get sufficient information for her procurement decision, she asks the vendors for the digital signatures C of the components which are usually delivered without the component code. Thereafter, the customer decrypts the signatures in order to get the corresponding ciphers which are transmitted[1] to the trust value manager. For each component two separate trust values are stored. One trust value is calculated according to the metric of Jøsang and Knapskog [20] while the other reflects the policy of Beth et al. [1]. Thus, more tolerant as well as more rigid customer policies are supported. Since a trust value consists of three values indicating the belief, disbelief, and uncertainty in a component (cf. Sec. 3), the trust value manager passes six values for each cipher component to the customer.

Now the customer can compare the trust values of the components in question. Besides procurement, they can also be used to decide about subjecting the component to a certification process as well as to determine the intensity of runtime monitoring measures (cf. Sec. 7). The trust values, however, describe only one aspect of a component. Other aspects comprise the price of a component, the interface structure which may facilitate or impede the integration of the component into the application, and quality of service issues like performance requirements. Moreover, one can also take the dependability of a component in consideration which can be calculated by similar metrics as the trust values [43].

In order to offer trust values with a high decree of certainty, a large number of expertises is needed. Therefore the trust value manager calls a customer about a week after an inquiry for trust values. The customer retransmits the cipher of the selected components, the results of customer-based certifications, as well as first evaluations of runtime monitoring. If the certification or the monitoring measures could not disprove that the behavior of a component complies with its contract description, the customer passes a positive rating. If the customer detected errors, she sends a negative expertise together with a log of interface actions proving the malfunctioning and preventing wrong accusations. Since certifications tend to be more thorough and profound than runtime monitoring, a customer-based certification is rated to three runtime evaluations. After the first

[1] In order to prevent manipulations of the ciphers, all data transmissions are protected by digital signatures.

inquiry, the trust value manager asks a customer every six weeks[2] about the experience with components gained by runtime monitoring. Moreover, if a component owner detects serious malicious component behaviors, she informs the trust value manager immediately.

Another valuable means of gaining expertises are certification authorities. If the component vendor or a customer initiates a third-party certification, the component is sent together with the digital signature C and the authority is asked to transmit the certification result to the trust value manager. Depending on the recommendation trust in the certification authority the certification results are calculated between three and 50 runtime monitoring expertises.

In order to protect application owners against malicious component behavior, the trust value manager offers an alarm service. A customer booking this service is immediately notified when bad expertises about a component are received. Thus, users can often isolate a faulty component before it causes harm for its application. Currently, two alarm modes are offered: In the first variant an alarm is only triggered if an user reports a severe security violation while in the second mode the alarm message is executed after any kind of negative evaluation.

7 Trust-Based Runtime Monitoring

The process to gain evaluations based on runtime monitoring can be facilitated by application of the security wrappers introduced in [17]. Here, a component in question is wrapped with special components checking the events at the component interface for compliance with security objectives fixed in the component contract. The constraints are modeled by state-transition-systems which are simulated during runtime (cf. [3]). Each model constrains the passing of events and the component interface in order to guarantee a certain security aspect. For instance, assume that the component in question A may only call a method M of a partner component B, if it previously received a certain credential from a third component C by means of C calling the method N of A. The corresponding constraint specification uses the states s and t. s is initial and states that the credential did not arrived yet while t models that the credential is available. The transitions concern the events M and N. When N is executed, the

[2]In case of electronic trust managers (cf. Sec. 7) the calls are performed every 48 hours.

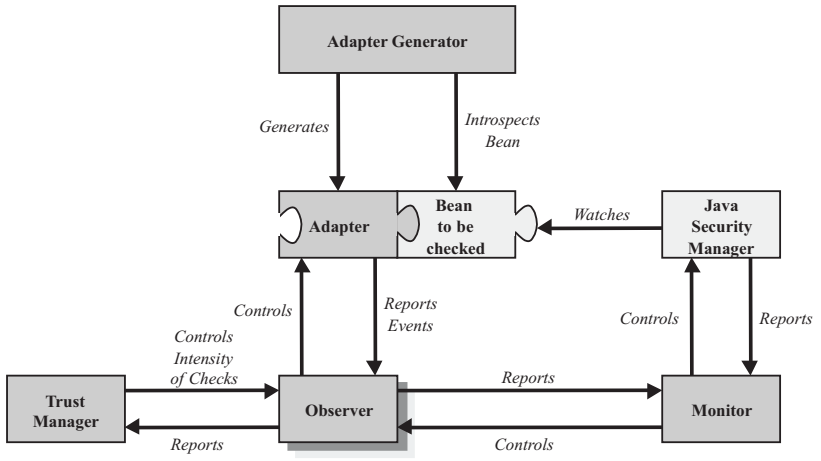

Figure 3: Security Wrapper Architecture

specification sets the current state to t. Transitions of event M state that this event may only be fired in state t but not in s. The particular model does not reflect other events at the interface.

The task of the security wrapper is to simulate the model and to check it for compliance with the real interface behavior. In particular, it stores and calculates the current state and compares interface events with the model. If in our example component A wants to executed the method call M, the wrapper first checks if the call is performed in state s or t. The method call is legal if the model simulation is in the state t and the wrapper passes M to component B. However, the call of method M in state s indicates that A tried the call without the credential which is a violation of the corresponding security objective. Here, the wrapper seals the component by blocking the interaction between the component and its environment. Moreover, the application operator is informed about the violation.

We specify the security constraints in the formal specification technique cTLA [16] which is based on Lamport's Temporal Logic of Actions (TLA) [25]. cTLA facilitates modeling of safety, liveness, and realtime [15] properties in a process style similar to high-level programming languages. Moreover, it supports the definition of constraint-oriented specifications (cf. [40]) each modeling a separate security aspect of a component. The security objectives can also be specified as diagrams in the popular Unified Modeling Language (UML) [7] and translated to cTLA processes (cf. [12]).

A first prototype implementation [28] is realized as a component structured system itself based on Java Beans [37]. Fig. 3 delineates the enforcement system. Each bean to be checked is wrapped by a special *Adapter* component. It does not interact directly with its environment but only via the adapter. Thus, incoming and outgoing events can be observed and — in case of sealing the bean — also blocked.

The compliance checks are performed by the *Observer* components. For each cTLA constraint specification a separate observer is generated which simulates a cTLA process and checks if the real bean behavior corresponds to the simulated behavior. Therefore the adapter forwards all interface events to the observers which perform the compliance checks and send the results back to the adapter. If all checks are correct, the adapter passes incoming events to the bean and outgoing events to the environment. On the other hand, if an observer detects a violation, the adapter blocks all ongoing events leading to the isolation of the bean in the application.

Other objects of the wrapper comprise an *Adapter Generator* analyzing a bean to be checked by introspection and creating an adapter based on this analysis. The *Monitor* acts as an interactive interface to the application administrator. If an observer detects a violation, it reports it to the monitor which shows it on the administrator screen. Moreover, the tool utilizes the built-in *Java Security Manager*. In order to prevent hidden data channels, the administrator may reduce the access of beans to certain system resources by parameterizing the security manager.

The *Trust Manager* is of particular interest for the work introduced in this paper since it represents the link between the security wrapper and the trust information system outlined above. If an observer detects a violation, it notifies not only the monitor but also the trust manager. If the trust manager considers the violation as severe, it sends a warning message to the trust value manager which includes a log of the events at the bean interface. Moreover, the trust manager replies inquiries of the trust value manager. If it was notified by an observer about a violation, it transmits a negative expertise and, otherwise, a positive evaluation report. Furthermore, the trust manager reacts on incoming alarm messages from the trust information system. After an alarm it calls an observer which causes the adapter to seal the bean in order to protect the application.

Experiences showed that the performance penalty of the security wrapper is less than 5 % on a state-of-the-art PC (cf. [17]). Nevertheless, one can also use the system to adapt the monitoring expenditure to the current trust value of a bean. Therefore, the trust manager implements a security policy stating the amount of observation with respect to a certain trust policy. In intervals, it inquires the trust information system for trust values. Based on the current values it can omit compliance tests by stopping observers. Here, two possibilities are available: At first, an observer can be removed which renders maximum performance gain but rules out the possibility to continue observations at a later time since the current state of the cTLA process is lost. To prevent this, in the second alternative an observer is switched into a special mode where it omits compliance checks but still calculates the current cTLA process states. Thus, the observer can be reactivated again if the trust values are getting worse. Moreover, it is possible to perform spot checks where an observer perform compliance tests only in intervals. The duration of the checks may be adjusted according to the trust value of the component.

8 Conclusion

We proposed an approach to support secure procurement and deployment of components by introduction of a trust information service employing the experience gained by other component users. Currently, a first prototype is under development [45]. It is created based on Java Beans and will be compatible with the security wrapper introduced in [17]. For a commercial realization, however, more problems have to be solved. In particular, financing issues have to be considered. Potential sources of income to finance the trust information service comprise the component vendors who may use good ratings for marketing, the component customers who are supported to buy secure components, and internet advertisements offered in the service web representation.

Another problem is to find solutions to protect component vendors against erroneous accusations causing bad trust values of a component wrongly. Here, trusted third-party components may be used to mediate conflicts between principles. For instance, an arbiter component may be called if a customer or a certification authority passes a bad evaluation to the trust value manager. The arbiter checks logs and audits of the event in question as well as the trust value of the component vendor and decides if the accusation is correct. The trust value manager may only recognize the negative expertise if the component vendor is pronounced guilty by the arbiter. The mediation can be supported by another class of components acting as witnesses. On behalf of the vendor, the customer, or the arbiter these witnesses may observe components and the incorporating ap-

plications. In case of an accusation, the witnesses support the arbiter by telling their observations. Furthermore, the trust information service can be extended in order to store evaluations and trust values not only of components but also of component vendors and customers. Thus, the arbiter may utilize the reputations of a vendor and a customer in order to decide about accusations.

References

[1] T. Beth, M. Borcherding, and B. Klein, "Valuation of Trust in Open Networks", in: *European Symposium on Research in Security (ESORICS)*, Brighton, LNCS 875, Springer-Verlag, 3–18 (1994).

[2] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins, "Making Components Contract Aware", *IEEE Computer*, vol. 32, no. 7, 38–45 (1999).

[3] J. Biskup and C. Eckert, "About the enforcement of state dependent security specifications", in: *Database Security*, eds. T. Keefe and C. Landwehr, Elsevier Science (NorthHolland), 3–17 (1994).

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis, "The KeyNote Trust Management System, Version 2", in: *Report RFC-2704*, IETF (1999).

[5] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis, "The Role of Trust Management in Distributed Systems Security", in: *Internet Programming: Security Issues for Mobile and Distributed Objects*, eds. J. Vitek and C. Jensen, Springer-Verlag (1999).

[6] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management", in: *17th Symposium on Security and Privacy*, Oakland, IEEE Computer Society Press, 164–173 (1996).

[7] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley Longman (1999).

[8] Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss, "REFEREE: Trust Management for Web Applications", *World Wide Web Journal*, vol. 2, 127–139 (1997).

[9] W. Farmer, J. Guttman, and V. Swarup, "Security for Mobile Agents: Issues and Requirements", in: *19th National Information Systems Security Conference (NISSC 96)*, 591–597 (1996).

[10] T. Fraser, L. Badger, and M. Feldman, "Hardening COTS Software with Generic Software Wrappers", in: *1999 IEEE Symposium on Security and Privacy* (1999).

[11] L. Gong, "Java Security Architecture (JDK1.2)", available via WWW: java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html (1998).

[12] G. Graw, P. Herrmann, and H. Krumm, "Constraint-Oriented Formal Modelling of OO-Systems", in: *Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS 99)*, Helsinki, Kluwer Academic Publisher, 345–358 (1999).

[13] J. Grundy, "Storage and retrieval of Software Components using Aspects", in: *2000 Australasian Computer Science Conference*, Canberra, IEEE Computer Society Press (2000).

[14] P. Herrmann, "Information Flow Analysis of Component-Structured Applications", to appear in: *17th Annual Computer Security Applications Conference (ACSAC'2001)*, New Orleans, ACM SIGSAC, IEEE Computer Society Press (2001).

[15] P. Herrmann and H. Krumm, "Formal Hazard Analysis of Hybrid Systems in cTLA", in: *18th IEEE Symposium on Reliable Distributed Systems (SRDS'99)*, Lausanne, IEEE Computer Society Press, 68–77 (1999).

[16] P. Herrmann and H. Krumm, "A Framework for Modeling Transfer Protocols", *Computer Networks*, vol. 34, no. 2, 317–337 (2000).

[17] P. Herrmann and H. Krumm, "Trust-adapted enforcement of security policies in distributed component-structured applications", in: *6th IEEE Symposium on Computers and Communications*, Hammamet, IEEE Computer Society Press, 2–8 (2001).

[18] A. Jøsang, "The right type of trust for distributed systems", in: *UCLA conference on New security paradigms workshops*, Lake Arrowhead, ACM, 119–131 (1996).

[19] A. Jøsang, "An Algebra for Assessing Trust in Certification Chains", in: *Network and Distributed Systems Security Symposium (NDSS'99)*, ed. J. Kochmar, The Internet Society (1999).

[20] A. Jøsang and S. J. Knapskog, "A metric for trusted systems", in: *21st National Security Conference*, NSA (1998).

[21] G. Karjoth, D. Lange, and M. Oshima, "A Security Model for Aglets", *IEEE Internet Computing*, 68–77 (1997).

[22] K. Khan, J. Han, and Y. Zheng, "Specifying security requirements and assurances of software components", in: *Australian Workshop on Requirements Engineering*, Brisbane, 57–65 (2000).

[23] K. Khan, J. Han, and Y. Zheng, "A Framework for an Active Interface

to Characterise Compositional Security Contracts of Software Components", in: *Australian Software Engineering Conference (ASWEC'01)*, Canberra, IEEE Computer Society Press, 117–126 (2001).

[24] R. Khare and A. Rifkin, "Weaving a Web of Trust", *World Wide Web Journal*, vol. 2, no. 3, 77–112 (1997).

[25] L. Lamport, "The Temporal Logic of Actions", *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 3, 872–923 (1994).

[26] U. Lindqvist and E. Jonsson, "A Map of Security Risks Associated with Using COTS", *IEEE Computer*, vol. 31, no. 6, 60–66 (1998).

[27] C. Lüer and D. S. Rosenblum, "WREN — An Environment for Component-Based Development", *Technical Report #00-28*, University of California, Irvine, Department of Information and Computer Science (2000).

[28] A. Mallek, "Sicherheit komponentenstrukturierter verteilter Systeme: Vertrauensabhängige Komponentenüberwachung" (in German), Diploma Thesis, University of Dortmund, Computer Science Department, 44221 Dortmund, Germany (2000).

[29] Microsoft, "The Microsoft COM Technologies", available via WWW: http://www.microsoft.com/com/comPapers.asp (1998).

[30] B. P. Miller, L. Fredrikson, and B. So, "An Empirical Study of the Reliability of Unix Utilities", *Communications of the ACM*, vol. 32, no. 12, 32–44 (1990).

[31] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", FIPS, edition 186 (2000).

[32] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)", FIPS, edition 180-1 (2000).

[33] Object Management Group, "CORBA Component Model Request for Proposals" (1997).

[34] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Communications of the ACM*, vol. 21, no. 2, 120–126 (1978).

[35] M. Shepherd, A. Dhonde, and C. Watters, "Building Trust for E-Commerce: Collaborating Label Bureaus", in: *2nd International Symposium on Electronic Commerce Technologies (ISEC'2001)*, eds. W. Kou, Y. Yesha, and C. J. Tan, Hong Kong, LNCS 2040, Springer-Verlag, 42–56 (2001).

[36] Sun Microsystems, "Enterprise Java Beans Technology — Server Component Model for the Java Platform (White Paper)", available via WWW: java.sun.com/products/ejb/white_paper.html (1998).

[37] Sun Microsystems, "Java Beans Specification", available via WWW: java.sun.com/beans/docs/spec.html (1998).

[38] C. Szyperski, "Component Software — Beyond Object Oriented Programming", Addison-Wesley Longman (1997).

[39] K. A. Tee, "E-Commerce in an Era of Creative Destruction", available via WWW: www.alumni.nus.edu.sg/Alumnus/jul2000/ecom.html (2000).

[40] C. A. Vissers, G. Scollo, and M. van Sinderen, "Architecture and specification style in formal descriptions of distributed systems", in: *Protocol Specification, Testing and Verification (PSTV'VIII)*, eds. S. Agarwal and K. Sabnani, IFIP, Elsevier, 189–204 (1988).

[41] J. Voas, "A Recipe for Certifying High Assurance Software", in: *22nd International Computer Software and Application Conference (COMPSAC'98)*, Vienna, IEEE Computer Society Press (1998).

[42] J. Voas, G. McGraw, A. Ghosh, and K. Miller, "Glueing together Software Components: How good is your Glue?", in: *Pacific Northwest Software Quality Conference*, Portland (1996).

[43] J. Voas and J. Payne, "Dependability Certification of Software Components", *The Journal of Systems and Software*, vol. 52, no. 2–3, 165–172 (2000).

[44] R. Wabbe, S. Lucco, T. E. Anderson, and S. L. Graham, "Efficient software-based fault isolation", in: *14th Symposium on Operating System Principles*, ACM, 203–216 (1993).

[45] M. Zülch, "Ein Werkzeug zum Vertrauensmanagement komponentenbasierter Software" (in German), to appear as: Diploma Thesis, University of Dortmund, Computer Science Department, 44221 Dortmund, Germany (2001).

Peter Herrmann studied Computer Science at the University of Karlsruhe (diploma in 1990). Since then he works as a researcher in the Computer Networks and Distributed Systems Group of the Computer Science Department at the University of Dortmund (Ph.D. in 1997). His research interests include security aspects of distributed component-structured software as well as formal-based development of distributed applications and hybrid technical systems.