[**Publication 7**] Zheng Yan and Ronan MacLaverty, "Autonomic Trust Management in a Component Based Software System", In *Proceedings of the 3rd International Conference on Autonomic and Trusted Computing (ATC2006)*, LNCS Vol. 4158/2006, pp. 279-292, China, September 2006.

http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/11839569_27

Autonomic Trust Management in a Component Based Software System

Zheng Yan, Ronan MacLaverty

Nokia Research Center, Helsinki, Finland

{zheng.z.yan, ronan.maclaverty}@nokia.com

Abstract. Trust plays an important role in a software system, especially when, the system is component based and varied due to component joining and leaving. How to manage trust in such a system is crucial for embedded devices, such as mobile phones. This paper presents a trust management solution that can manage trust adaptively in a component based software (CBS) system. We develop a formal trust model to specify, evaluate and set up trust relationships that exist among system entities. We further present an autonomic trust management architecture that adopts a number of algorithms for trust assessment and maintenance during component execution. These algorithms use recent advances in Subjective Logic to ensure the management of trust within the CBS system.

1 Introduction

The growing importance of software in the domain of mobile systems introduces special requirements on trust. The first requirement is that any software design must support a product-line approach to system development. This normally implies that system software consists of a number of components that are combined to provide user features. Components interact over well defined interfaces; these are exported to applications that can combine and use the components to provide features to consumers. Thus, common components can be effectively shared by applications. A typical feature of mobile devices with CBS is to allow addition of components after deployment, which creates the need for run-time trust management.

From system point of view, trust is the assessment of trustor on how well the observed behavior (quality attributes) of trustee meets trustor's own standards for an intended purpose [1]. From this, the critical characteristics of trust can be summarized, it is: subjective, different for each individual in a certain situation; and dynamic, sensitive to change due to the influence of many factors. Therefore, we need a proper mechanism to support autonomic trust management not only on trust establishment, but also on trust sustaining.

Most trust management systems focus on protocols for establishing trust in a particular context, generally related to security requirements. Others make use of a trust policy language to allow the trustor to specify the criteria for a trustee to be considered trustworthy [2]. Grandison and Sloman studied a number of existing trust man-

agement systems in [2]. These systems evaluate a viewpoint of trust that is quite closely tied to systems that implement access control or authentication. In [3], SULTAN framework provides the means to specify, and evaluate trust relationships for Internet applications. However, its computational overhead means it is not feasible inside an embedded system, especially in the role of trust monitor and controller.

The evaluation of trust, only when a relationship is set up, does not cater for the evolution of the relationship from its stated initial purpose to accommodate new forms of interaction. Furthermore, the focus of the security aspect of trust tends to assume that the other non-functional aspects of trust, such as availability and reliability, have already been addressed.

At present, there is no common framework to enable trust management in a commercial CBS system, even though there is a pressing need to support a range of new applications. This framework must support autonomic trust management through trust assessment and maintenance over the dynamic CBS system, consisting of different functionalities provided by various disparate companies. We need technologies for the development and validation of the trusted systems based on the integration of multi-party software while at the same time reducing the cost and integration time. The research described in this paper is an initial attempt at addressing some of these needs, which aims to establish trustworthy middleware architecture for the embedded systems with CBS.

This paper introduces an autonomic trust management solution that supports dynamic changes of trustworthiness in the CBS system. The main contributions of this paper are: a formal trust model that specifies the trust relationships for the CBS systems or devices; an autonomic trust management architecture that incorporates the new trust model allowing explicit representing, assessing and ensuring of the trust relationships, and deriving trust decisions in a dynamic environment; and a range of algorithms for incorporating trust into a system, demonstrating the feasibility of the architecture.

The rest of the paper is organized as follows. Section 2 specifies the problems we need to overcome. Section 3 presents a formal trust model we will apply into the autonomic trust management in the CBS systems. Section 4 details the design of trust management framework and develops algorithms for the autonomic trust management. Finally, conclusions and future work are presented in Section 5.

2 Trust in CBS Systems

In mapping trust to the CBS systems we can categorize trust into two aspects: trust in the component, and trust in a composition of components. For the component-centered aspect we must consider trust at several decision points: at download time and during execution. At a component download time, we need to consider whether a software provider can be trusted to offer a component. Furthermore, we need to predict whether the component is trustworthy for installation. More necessarily, when the component is executed, we have to ensure it can cooperate well with other components and the system provides expected performance and quality. The trust relationship changes during the above procedure.

When discussing a CBS system, the execution of components in relation to other entities of the system needs to be taken into account. Even though the component is trustworthy in isolation, the new joined component could cause problems because it will share system resources with others. This influence will impact the trustworthiness of the system. Consequently, the system needs mechanisms to control its performance, and to ensure its trustworthiness even if internal and external environment changes. Additionally, some applications (e.g. a health care service) need special support for trust because they have high priority requirements, whereas game playing applications, while exhibiting similar functionality (e.g. a network connection) will not have the same priority. Therefore, system-level trustworthiness is dependent on the application domain, so the system needs a trust management framework that supports different trust requirements from the same or different components. This paper mainly focuses on the autonomic trust management for CBS at system runtime.

3 A Formal Trust Model

Trust involves two roles: a trustor (*tr*) and a trustee (*te*). In a CBS system, a trustor can be a system user or the user's representatives (e.g. a trusted computing platform and its protected trust management framework). A trustee is the entity who holds the essential competence and quality to offer the services the trustor expects. Since trust is subjective, the trust is greatly influenced by a trustor's policy (*py*). Trust is also dynamic and is related to the context (*ct*) of the trustor and the trustee, for example, time (*t*), environment (*e*) and the intended purpose (*p*). Most importantly, it is mainly influenced by the trustee's competence, performance and quality. Some quality attributes of a trustee (*qa*) behave as referents for trust assessment. A referent will be evaluated based on evidence (*ev*) collected from previous experience of the trustor or other entities (e.g. recommenders). The result of a trust assessment could be a value set (*b, d, u*) that reflects the trustor's opinion (*op*) on the trust. This opinion is described as a 3-tuple: belief, disbelief and uncertainty [4].

3.1 Definitions

A CBS system can be represented as a structure (E, R, O), where E represents the set of the system entities, R the set of trust relationships between the entities, O the set of operations for the management of such trust relationships.

1) CBS system entities:

These entities can be any parties that are involved into or related to the CBS system. These entities include a CBS system user, a component consumer, a component provider, a service, a component (composition of components), an application, a subsystem and a system, as well as an operation or a mechanism provided by the system.

An application is a software entity that provides a set of functions to a user. A component is a unit of trading that may contain multiple services. A service is a unit of software instantiation that is contained in a component and conforms to a component model. A system is a combination of a platform, a set of components, a runtime

environment (RE) and a set of applications that can provide a user with a set of functions. A platform provides access to the underlying hardware. The relationships among above entities are described in Figure 1.

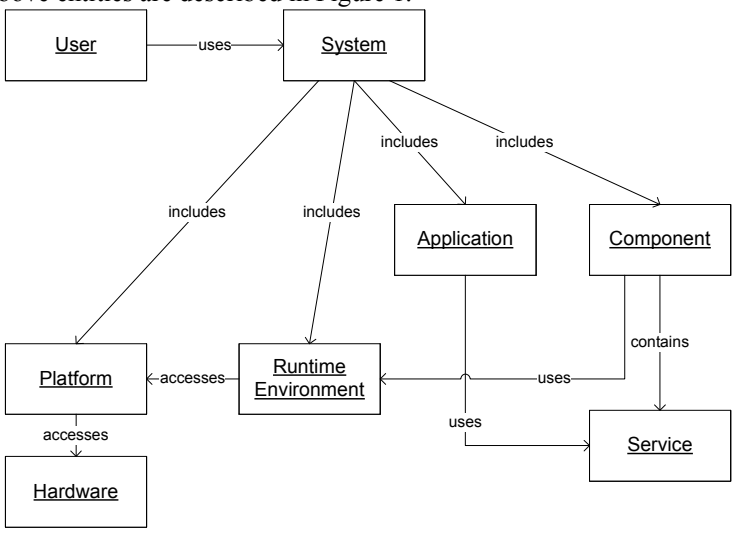

Figure 1: Relationships of CBS system entities

2) Trust relationship

A trust relationship in a CBS system can be specified as a 6-tuple $TR = \{tr, te, py, ct(t,e,p), ev, op(b,d,u)\}$ which asserts that entity *tr* trusts entity *te* with regard to *tr*'s trust policy *py* in the context *ct(t,e,p)*, based on the evidences of *te*: *ev*, and *op(b,d,u)* indicates the trust valuation.

Where:

- *tr* and *te* are subsets of the set of system entities(E).
- *py* is the subset of the set (PY) of all policies regarding trust management. $PY = \{to, ir, ep, cr\}$, where *to* is a threshold opinion for trust, $ir = \{ir_{qa\_1}, ir_{qa\_2}, ir_{qa\_3}, ..., ir_{qa\_n}\}$ is the importance rates of different quality attributes *qa* of the *te*, and *ep* is the policy about the evidence used for the trust assessment. *cr* ($cr = \{tv_{qa\_1}, tv_{qa\_2}, tv_{qa\_3}, ..., tv_{qa\_n}\}$) is the criteria for setting positive or negative points on different quality attributes (refer to section 4.3 for details). It specifies the trusted values or value scope of different factors reflecting the quality attributes of the trustee.
- *ct(t,e,p)* expresses the context of the trust relationship, in which *t* is the time constraint during which the relationship is valid, *e* is the environment of the trust relationship (e.g. system configurations and the domain the system located), and *p* is the purpose of establishing the trust relationship.
- *ev=(qa, rn)* denotes the evidences of quality attributes, where *rn* is a subset of recommendations RN , and *qa* is the subset of the set

$QA = \begin{Bmatrix} security(\text{confidentiality}, \text{integrity}, \text{safety}); reputation; usability; \\ dependability(\text{reliability}, \text{availability}, \text{adaptability}, \text{maintainability})... \end{Bmatrix}$. Each quality attribute has a number of factors that are referents of it. These factors can be quantified and thus monitored. For example, the 'availability' quality attribute can be reflected by uptime and response time. The trust adaptability can be measured as *MTTS/(MTTE+MTTS+MTTB),* where *MTTB* is the mean time of response regarding trust relationship break, *MTTS* is the mean time of trust sustaining and *MTTE* is the mean time for trust establishment and re-establishment. The *QA* has a tree-like structure, which benefits the reasoning of trust problems.

A recommendation can be expressed as $rn = (rr, re, te, op, ct)$, where it asserts that *rr* recommends *re* that *rr* holds opinion *op* on *te* in the context of *ct*. Note that $rn = (rr, re, te, op, ct)$ is one kind of evidence.

- *op(b,d,u)* is the trust assessment result of this trust relationship. It is probabilities of opinion on *te* regarding belief (*b*), disbelief (*d*) and uncertainty (*u*). Particularly, *b*, *d*, and *u* satisfy: $b + d + u = 1$, and $0 \le b, d, u \le 1$. Herein, belief means the probability of the entity *te* can be trusted by the entity *tr*; disbelief means the probability of *te* can not be trusted by *tr*; and uncertainty fills the void in the absence of both belief and disbelief. Particularly, the bigger the value of *b*, *d*, and *u*, the more probability.

3) Trust management operations

The trust management operations compose a set $O = \{T_e, T_m, T_a, T_c\}$, where T_e is the set of operations or mechanisms used for trust establishment and re-establishment, T_m is the set of operations or mechanisms applied for monitoring and collecting the evidences or the factors regarding the quality attributes of the trustee, T_a is the set of operations or mechanisms for trust assessment, and T_c is the set of operations or mechanisms for controlling trust in order to sustain the trust relationship. The operations in T_e and T_c are classified in terms of enhancing different quality attributes. Thus the system knows which operation should be considered in order to ensure or support some quality attribute.

3.2 Trust model for components

The trust model for a component, given below, describes the trust specifications of all the operations implemented by services in the component. The trust request level (*trust_level*) indicates the importance of the operation (*impl_opr*). This level is upgraded when an application requests a service. *Resource* and *consumption* specifies the resource requirements that the operation requires in order to provide the performance described by *performance*. In addition, composition rules (*com_rule*) are criteria for composing this model with other trust models. The composition rule could be as simple as selecting the maximum or minimum value, or as complicated as an algorithm based on the relationships of service operations. How to specify a composition rule is beyond the scope of this paper.

m = *TM,*

	where *m* is a *Trust Model* and *TM* is a set of *tm* (the trust specification of an operation).
tm	= (*impl_opr*, *resource*, *consumption, trust_level, performance, com_rule*), for operation *impl_opr.*
Resource	= *r* ∈ {*memory*, *cpu*, *bus*, *net*…}.
Consumption	= *claim,* in case *resource* is *cpu.*
Consumption	= (*claim*, *release*), in case *resource* is *memory.*
Consumption	= (*claim*, *time*), in case *resource* is *bus.*
Consumption	= (*claim*, *speed*), in case *resource* is *net.*
Trust_level	= *tl* ∈ *TL*, where *TL* is the set of trust level.
Performance	= (*attribute, value*) *Attribute = attr* ∈ {*response time, uptime, mean time of failure, mean time of hazard, mean time of repair,…*}: a set of factors that are referents of the quality attributes. *Value = asserted trust value of the attribute*
Com_rule	= *crule* ∈ *CRULE*, where *crule* is a composition rule and *CRULE* is a set of *crule* (composition rules for composing with other trust models)
Crule	= (*cr_resource, cr_consumption, cr_trust_level, cr_performance, composition_type*), specifying composition rules for *resource*, *consumption*, *trust_level* and *performance*.

The trust model of a component can be composed based on the composition rules. It has several usages. At download time, it can be used to help system trust management framework to predict whether a component may have some trust influence on the system. The system firstly composes all related trust models based on their composition rules, and then investigates if the resource and trust requirements can be satisfied. At execution time, the trust model is used by the system execution framework to arrange resources for the services. In addition, it could help the trust management framework to monitor the performance of the services (e.g. the composed performance could play as the default trust criteria for the trust assessment), thus assess if the component's services and the subsystem containing the component are trusted or not. It could also be used to reason the trust problems at some services in a component.

3.3 Autonomic trust management model

The trust assessment can be expressed as a 6-tuple $TA = (tr, te, ct, ev, py, TM)$, which asserts that *tr* assesses *te*'s trustworthiness in the context *ct*, based on the *tr*'s policy *py* and according to evidence *ev* on the *te*'s quality and the trust model *TM*. The trust assessment result has three possibilities: 1 – trusted; 0 – distrusted; and -1 – unknown. Autonomic trust management can be expressed as a 3-tuple $ATM = (TR, TA, O)$, which asserts that operations *O* are applied for the trust relationship *TR* according to the trust assessment *TA*.

4 An Autonomic Trust Management Architecture

4.1 CBS system structure

The software architecture of a CBS system has many constraints; most of these are common across a range of industries and products. Therefore, although each product has a different software structure, a general architectural style has emerged that combines component based development with product-line software architectures [5]. The work here is based on the Robocop Component model [6], which embodies this generic style. The style consists of layered development architecture with 3 layers: an application layer that provides features to a user; a component-based middleware layer that provides functionality to applications; and, a platform layer that provides access to lower-level hardware. Using components to construct the middleware layer divided this layer into two developmental layers: a component sub-layer that contains a number of executable components and a runtime environment (RE) sub-layer that supports component development.

The component runtime supporting frameworks also exist at the runtime sub-layer. These provide functionalities for supporting component properties and for managing components. These frameworks also impose constraints on the components, with regard to mandatory interfaces, associated metadata etc. The runtime environment is consists of a component framework that treats DLL-like components. This provides a system-level management of the software configuration inside a device. Each component contains services that are executed and used by applications. The services have interactions with other services; they consume resources; and, they have trust models as described in 3.2 attached.

For some of the frameworks in the runtime environment, they have to be supported with platform functionality. For example, for resource framework, support for resource usage accounting and enforcement is required from the platform layer. In terms of trust management, the platform needs to provide security mechanisms, such as access control, memory protection and encryption/decryption. In this case the security framework offers functionality for the use of security mechanisms, provided by the platform, to develop and maintain a secure system. The platform layer also provides trusted computing support on the upper layers [7, 8].

Placing trust management inside this architecture means linking the trust management framework with other frameworks responsible for the component management (including download), the security management, the system management and the resource management. Figure 2 describes interactions among different functional blocks inside the running environment sub-layer. The trust management framework is responsible for the assessment on trust relationships and trust management operations, system monitoring and autonomic trust managing. The download framework requests the trust framework for trust assessment about components to decide if to download a component and which kind of mechanisms should be applied to this component. When a component service needs cooperation with other components' services, the execution framework will be involved, but the execution framework will firstly request the trust management framework for decision. The system framework takes care

of system configurations related to the components. The trust management framework is located at the core of the runtime environment sub-layer. It monitors the system performance and instructs the resource framework to assign suitable resources to different processes. This allows the trust management framework to shutdown any misbehaving component, and to gather evidence on the trustworthiness of a system entity. Similarly, the trust management framework controls the security framework, to ensure that it applies the necessary security mechanisms to maintain a trusted system. So briefly, the trust management framework acts like a critical system manager, ensuring that the system conforms to its trust policies.

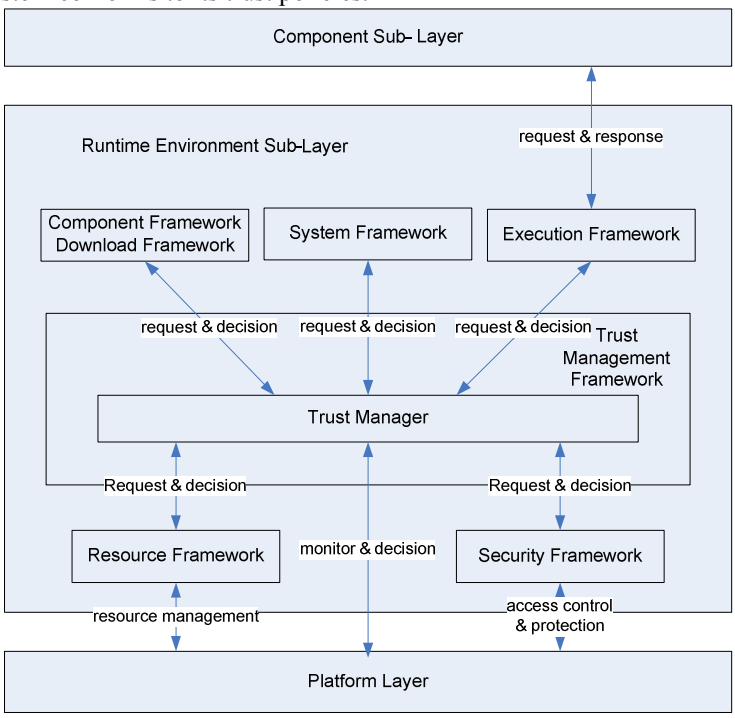

Figure 2: Relationships among trust framework and other frameworks

4.2 Trust management framework

Figure 3 illustrates the structure of the trust management framework. The trust manager is responsible for trust assessment and trust related decision-making, it closely collaborates with the security framework to offer security related management. The trust manager is composed of a number of functional blocks:

- Trust policy base saves the trust policy (*py*) regarding making trust assessment and decision.
- Recommendation base saves various recommendations.
- Experience base saves the evidence *Qa(ev)* collected from the CBS system itself in various contexts;

- Decision/reason engine is used to make trust decision by request. It combines information from experience base, recommendation base and policy base to conduct the trust assessment. It is also used to identify causes of trust problems.
- Mechanism base saves opinions regarding the mechanisms in T_e and T_c that are supported by the system and attached to special context or configurations.
- Selection engine is used to select suitable mechanisms to ensure the system's trust in a special context.

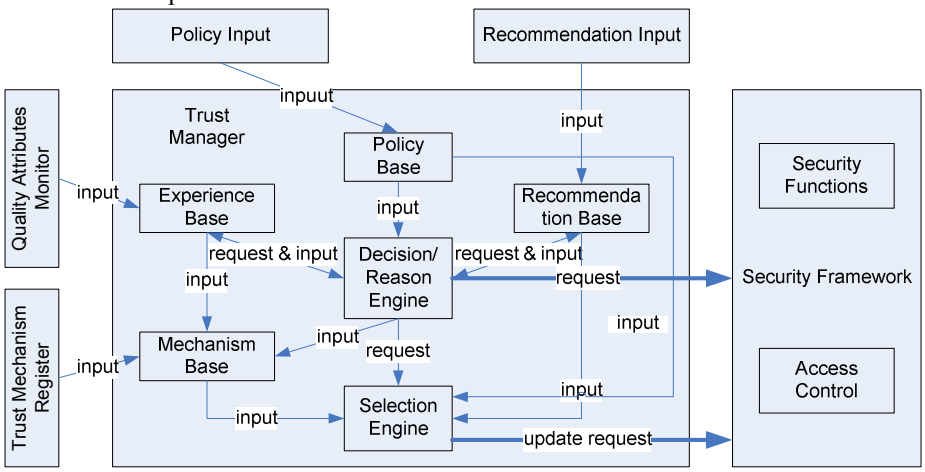

Figure 3: A structure of trust management framework

In addition, recommendation input is the interface for collecting recommendations. Policy input is the interface for the CBS system entities to input their policies. Trust mechanism register is the interface to register trust mechanisms that can be applied in the system. Quality attributes monitor is the functional block used to monitor the CBS system entities' performance regarding those attributes that may influence the trust. The trust manager cooperates with other frameworks to manage the trust of the whole system.

4.3 Trust assessment at runtime

There are several existing mechanisms that can be applied for assessing trust through the evidence. Here subjective logic (SL) [4] has been chosen as the formal base for trust assessment because of its sound mathematical foundation in dealing with evidential beliefs; and the inherent ability to express uncertainty explicitly. Subjective Logic consists of a belief model called opinion and set of operations for aggregating opinions. Herein, we apply a simplified scheme of the Subjective Logic as in [9, 10]. We use seven SL operators to illustrate how to assess trust based on the formal trust model and the trust management framework. (Refer to Appendix for Definition 5-7, and [9, 10] for Definition 1-4.)

Definition 1 (Bayesian Consensus) – operator $\oplus$: This operator can be used to combine opinions of different entities on the same entity together. For example, it can be

used to combine different entities' recommendations on the same entity (e.g. a service) together.

Definition 2 (Discounting) – operator $\otimes$: This operator can be used to generate opinion of a recommendation or a chain of recommendations. For example, it can be used by an entity (trustor) to generate an opinion on a recommendation.

Definition 3 (Conjunction) – operator $\wedge$: This operator can aggregate an entity's opinions on two distinct entities together with logical AND support.

Definition 4 (Disconjunction) – operator $\vee$: This operator can aggregate an entity's opinions on two distinct entities together with logical OR support.

Definition 5 (Adding) – operator $\sum$: This ad hoc operator can be used to combine the opinions on a number of the trustee's quality attributes. The combination is based on the importance rates of the attributes.

Definition 6 (Opinion generator) – operator θ: This operator is used to generate an opinion based on positive and negative evidence. Note that $op(x)$ stands for the opinion on x and $op(x.qa1)$ stands for the opinion on x's quality attribute *qa1*.

Definition 7 (Comparison) – operator $\geq_{op}$: This operator is used to compare two opinions, especially to decide if an opinion is over a threshold presented by another opinion and order a number of opinions.

At runtime, the quality attribute monitor monitors the trustee's performance with respect to its quality attributes. In the experience base, for each quality attribute, if the monitored performance is better than the criteria (saved in the policy base), the positive point of that attribute is increased by 1. If the monitored result is worse than the criteria, the negative point of that attribute is increased by 1. The opinion of each quality attribute can be generated based on the opinion generator θ. In addition, based on the importance rates of different attributes *ir*, combined opinion on the trustee can be calculated by applying the operator $\sum$. By comparing to the trust threshold opinion (*to*), the decision engine can decide if the trustee is still trusted or not. The algorithm for trust assessment at runtime is described as below.

Initialization

te: the assessed target (a system or subsystem or a service)

py(to, ir, ep, cr): the policy on *te*:

$n_{qa\_i} = p_{qa\_i} = 0$; $r_{qa\_i} = 2$; $(i = 1,...,n)$

op(qa_i) = (0,0,1); op(te) = (0,0,1)

1. Monitor *te*'s performance regarding *te*'s quality attributes in specified period *t*.
2. For $\forall qa\_i (i = 1,...,n)$,

   If the monitored result is better than $py.cr.tv_{qa\_i}$, $p_{qa\_i}++$;

   Else, $n_{qa\_i}++$
3. For $\forall qa\_i (i = 1,...,n)$, calculate the opinion:

   $op(qa\_i) = \theta(p_{qa\_i}, n_{qa\_i}, r_{qa\_i})$.

4. Based on the importance rates on different attributes, calculate a combined opinion: $op(te) = \sum_{i=1}^{n} \{ir_{qa\_i}, op(qa\_i)\}$.

5. If $op(te) \geq_{op} py.to$, make trust decision; else, make distrust decision.

4.4 Algorithm for autonomic trust management

Autonomic trust management includes several aspects.

- Trust establishment: the process for establishing a trust relationship between a trustor and a trustee. The trust establishment is required when a component or a bundle of components is downloaded and installed at the system.
- Trust monitoring: the trustor or its delegate monitors the performance of the trustee. The monitoring process aims to collect useful evidence for the trust assessment.
- Trust assessment: the process for evaluating the trustworthiness of the trustee by the trustor or its delegate. The trustor assesses the current trust relationship and decides if this relationship is changed or not. If it is changed, the trustor will make decision which measure should be taken.
- Trust control and re-establishment: if the trust relationship will be broken or is broken, the trustor will find reasons and take some measure to control or re-establish the trust.

Trust management operations applied by the system are registered at the mechanism base with its attached context *ct*. The opinions on all trust management operation are generated at the mechanism base. Based on a trust assessment result, if the result is trusted, increase the positive point of the applied operations by 1. If the result is distrusted, the trust manager reasons the problem based on the structure of *QA* and finds out the operations that cannot ensure a trust. At the mechanism base, the system increases the negative point of those operations by 1. The opinions on all applied trust management operations can be generated based on the opinion generator θ. If the opinions on some operations are below threshold, or the trust assessment result is not trusted, the operations that raise problems should be upgraded or replaced by better ones. At the selection engine, we select suitable operations based on the following mechanism. For each composition of a set of operations, we generate a common opinion on it through combining the opinion of each operation. If the combined opinion is above the threshold, we save it. By ordering all possible compositions, we can select one composition with the highest opinion belief via applying operator $\geq_{op}$. The algorithm for the trust assessment on the operations and the operation selection is described below.

Initialization

Considering ATM = (TR,TA,AO)

$AO = \{ao\_i\}(i = 1,...n)$: the trust management operations applied for *TR*

$n_{ao\_i} = p_{ao\_i} = 0$; $r_{ao\_i} = 2$; $(i = 1,...,n)$
op = op(ao_i) = (0,0,1)
py(to, ir, ep, cr): the policy on *AO*
$S = \phi$: the set of selected operations

At the Mechanism Base, generate opinions on operations

1. Do trust assessment, if $TA = 1$, $p_{ao\_i} + + (i = 1,...,n)$
   Else, find the operations *po_i* that cause the problems:
   $PO = \{po\_i\}(i = 1,...,k)$ $(PO \subseteq AO)$, $n_{ao\_i} + + (i = 1,...,k); ao\_i \in PO$
2. For $\forall ao\_i$, $op(ao\_i) = \theta(p_{ao\_i}, n_{ao\_i}, r_{ao\_i})$
3. If $py.to \geq_{op} op(ao\_i)$ or $TA \neq 1$, put *ao_i* (with opinion below threshold) or *po_i* into a set $RO = \{ro\_i\}(i = 1,...,l)$, upgrade these operations in *RO* with better ones
   At the Selection Engine, select suitable operations.
   For each composition of a set of upgrading operations CO, do
   3.1 Get existing opinions of new selected operations supported by the system *op(co_j)* (j=1,...,m)
   3.2. Aggregate above opinions
   $op = \wedge\{op(co\_j), op(ao\_i)\}(j = 1,...,m)(i = 1,...,n-l)$, $ao\_i \in AO - RO$
   3.3. If $op \geq_{op} py.to$, add *CO* into *S*
   3.4. If $S \neq \phi$, order all opinions of the operation sets in S using $\geq_{op}$, select operation set with highest opinion belief from S; else raise warning message
4. Go to step 1

5 Conclusions and Future Work

This paper is the first to develop a trust management solution for the CBS system based on the Subjective Logic. We have identified the trust issues in the CBS system. We then developed a formal trust model to specify, evaluate and set up trust relationships amongst system entities. Based on this trust model, we further designed the autonomic trust management architecture to overcome the specified issues. This design is compatible with the CBS system architecture. Thus it can be easily deployed in practice in order to enhance the trustworthiness of the middleware software. Once instantiated this architecture allows explicit trust policies to be defined and managed, thereby it supports human device interaction and provides autonomic trust management with the guideline of the system users. In addition, the proposed trust management architecture will enable the trust for both system users and system internal entities since it supports managing the trust relationship between any two entities inside the CBS system.

Desirable emerging properties can be obtained by applying this proposed trust management architecture to a CBS device. These include enabling the trust assessment at runtime based on the system monitoring on a number of quality attributes of the assessed entity; autonomic trust management on the basis of trust assessment and auto-selection of trust management operations. These emerging properties allow the trust at the system runtime to be better addressed.

For future work, we will further refine the architecture design, improve the algorithms towards context-aware support and study the adaptability of this trust management solution. We are planning to build and test the trust management architecture inside a mobile device, to check its feasibility and to gain experience in the practical use for the protection of the mobile device from both malicious attacks and unforeseen feature interactions. The goal of this is to ensure that users have a positive relationship with their devices.

Acknowledgement

This work is sponsored by EU ITEA Trust4All project. The authors would like to thank Dr. Peng Zhang, Nokia Venture Organization, Dr. Silke Holtmanns, Nokia Research Center, and Dr. Rong Su, Technische Universiteit Eindhoven, for their valuable comments.

References

1. DE Denning, A New Paradigm for Trusted Systems, Proceedings of the IEEE New Paradigms Workshop, 1993.
2. T. Grandison and M. Sloman, A Survey of Trust in Internet Applications, IEEE Communications and Survey, Forth Quarter, 3(4), pp. 2–16, 2000.
3. Tyrone Grandison, Morris Sloman, Trust Management Tools for Internet Applications. In Proceedings of the First International Conference of Trust Management (iTrust 2003), Crete, Greece, May 2003.
4. A. Jøsang, A Logic for Uncertain Probabilities, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 9(3), pp.279–312, June 2001.
5. Van Ommering, R. 2002. Building Product Populations with Software Components. In Proceedings of the 24th International Conference on Software Engineering, ICSE '02, Orlando, Florida, May 19 - 25, 2002, ACM Press, New York, NY, pp. 255-265.
6. Muskens, J. and Chaudron, M. 2004. Integrity Management in Component Based Systems. In Proceedings of the 30th EUROMICRO Conference (Euromicro'04), Volume 00, Washington, DC, August 31 - September 03 2004, pp. 611-619
7. England Paul, Lampson Butler, Manferdelli John, Peinado Marcus, Willman Bryan, A Trusted Open Platform, IEEE Computer Society, July 2003, pp. 55-62.
8. Trusted Computing Group (TCG), TPM Specification, version 1.2, 2003. https://www.trustedcomputinggroup.org/specs/TPM/
9. Lin, C.; Varadharajan, V.; Wang, Y.; Pruthi, V., Enhancing Grid Security with Trust Management**,** Proceedings of IEEE International Conference on Services Computing (SCC 2004), 15-18 Sept. 2004, pp. 303 – 310.

10.Twigg, A., A Subjective Approach to Routing in P2P and Ad Hoc Networks. In Proceedings of the First International Conference of Trust Management (iTrust 2003), Crete, Greece, May 2003.

Appendix: Definitions of Additional Subjective Logic Operators

Definition 5 (adding) – Operator $\sum$

Let $QA = \{qa_1, qa_2, ..., qa_n\}$ is a set of attributes that may influence an entity's opinion on a proposition. $IR = \{ir_1, ir_2, ...ir_n\}$ ($\sum_{i=1}^{n} ir_i = 1$) is a set of importance rates of attributes in *QA*, which marks the entity's weight of considerations on different attributes. That is ir_i is the importance rate of qa_i. $\Omega = \{\omega_1, \omega_2, ...\omega_n\}$ is a set of opinions about quality attributes. $\omega_i = (b_i, d_i, u_i)$ is the entity's opinion on qa_i. Let $\omega_\Sigma = (b_\Sigma, d_\Sigma, u_\Sigma)$ be opinion such that

$$b_\Sigma = \sum_{i=1}^{n} ir_i b_i \; ; \; d_\Sigma = \sum_{i=1}^{n} ir_i d_i \; ; \; u_\Sigma = \sum_{i=1}^{n} ir_i u_i$$

Then ω_Σ is called the sum of ω_i on a proposition.

Definition 6 (opinion generator) – Operator θ

Let $\omega = (b, d, u)$ be an entity's opinion about a proposition (or its attributes). Where

$b + d + u = 1$

$b = p/(p + n + r)$

$d = n/(p + n + r)$

$u = r/(p + n + r)$

and *p* is the positive points of evidence on the proposition, *n* is negative points of evidence on the proposition, $r \geq 1$ is a parameter controlling the rate of loss of uncertainty, which can be used to tune the use of uncertainty in the model for the requirements of different scenarios (we often take $r = 2$). Note that other definitions on *b, d* and *u* can be also applied.

This operator $\theta(p, n, r)$ can be used for generating an opinion based on positive and negative evidence.

Definition 7 (Comparison) - Operator $\geq_{op}$

Given two opinions $\omega_A(b_A, d_A, u_A)$ and $\omega_B(b_B, d_B, u_B)$, we define $\geq_{op}$ as an opinion comparison operator, whereby $\omega_A \geq_{op} \omega_B$ holds, if $b_A > b_B; d_A < d_B; u_A < u_B$. And we say that opinion $\omega_A(b_A, d_A, u_A)$ is over a threshold presented by $\omega_B(b_B, d_B, u_B)$.