2013

# Composition and combination-based object trust evaluation for knowledge management in virtual organizations

Yanjun Zuo
*University of North Dakota*, yanjun.zuo@und.edu

Brajendra Panda

# Composition and Combination-Based Object Trust Evaluation for Knowledge Management in Virtual Organizations

**Classification:** Research paper

**Purpose -** This paper aims to develop a framework for object trust evaluation and related object trust principles to facilitate knowledge management in a virtual organization. It proposes systematic methods to quantify the trust of an object and defines the concept of object trust management. The study aims to expand the domain of subject trust to object trust evaluation in terms of (1) whether an object is correct and accurate in expressing a topic or issue and (2) whether the object is secure and safe to execute (in the case of an executable program). By providing theoretical and empirical insights about object trust composition and combination, this research facilitates better knowledge identification, creation, evaluation, and distribution.

**Design/methodology/approach -** This paper presents two object trust principles – trust composition and trust combination. These principles provide formal methodologies and guidelines to assess whether an object has the required level of quality and security features (hence it is trustworthy). The paper uses a component-based approach to evaluate the quality and security of an object. Formal approaches and algorithms have been developed to assess the trustworthiness of an object in different cases.

**Findings –** The paper provides qualitative and quantitative analysis about how object trust can be composed and combined. Novel mechanisms have been developed to help users evaluate the quality and security features of available objects.

**Originality/value** – This effort fulfills an identified need to address the challenging issue of evaluating the trustworthiness of an object (e.g., a software program, a file, or other type of knowledge element) in a loosely-coupled system such as a virtual organization. It is the first of its kind to formally define object trust management and study object trust evaluation.

## 1. INTRODUCTION

In the past decade, the formation of virtual organizations (VOs) has become a global trend for organizations to work cooperatively to remain competitive in a market-driven environment. There are growing economic motivations to spread and share information and other forms of intellectual items among the participants of a VO. The ability to analyze information faster and more efficiently than before gives the organizations advantages to reduce operation costs and to seize opportunities for business prospects. The power of information sharing strengthens organizations and has led to the science of knowledge management.

In our discussion, a VO has a broad scope and includes any kind of loosely coupled virtual communities with independent participants using information systems to share and exchange resources (e.g., data, software, or other knowledge elements). The participants of a VO are self-organized and goal-oriented towards some common interests. Consider an open source software community such as GNUe (Elliott & Scacchi, 2004) - no lead organization or prime contractor has brought together the alliance in such a network. The software developers voluntarily contribute to the development of large-scale software. Such a VO is so decentralized that no company or corporate executive has sole administrative authority or resource control to determine what work will be done, what the schedule will be, and who will be assigned to perform specified tasks (Scacchi, 2007). The participants self-organize their activities and share their work to achieve the project goals. As such, the scope of a VO is often large enough to cross company boundaries. In addition to open source software communities, other types of VOs include cloud computing communities (Ristenpart, *et al.* 2009; Cody, *et al.* 2008), electronic marketplaces (Son & Benbasat, 2007; Soh, *et al.* 2006), online social networks (Perer, *et al.* 2006; Gross, *et al.* 2005) and virtual world communities (Bray, *et al.* 2007; Clemons, *et al.* 2006; Dellarocas, 2003).

In this paper, we study how the VO participants evaluate and share intelligent items (called objects) such as software programs (in an open source software community) or business data (in a business partner network). Creating, evaluating, sharing, and adopting those knowledge elements are crucial for an organization to leverage knowledge globally and maintain a competitive advantage. However, knowledge quality assurance is a challenging undertaking in such a loosely coupled system due to the open and dynamic nature of a VO. There is no rigid procedure for every participant to follow in knowledge/information processing. The standards and techniques that the participants have adapted could vary widely. Different entities may have different knowledge/information management cultures and processing environments. Therefore, there is a lack of rigorous standards and organization-wide central management to ensure object quality. Data or knowledge from other parties could be incomplete, obsolete, or incorrect. Consider an open source software community. The number of open source projects is vast. Projects range from low-quality individual efforts to high quality enterprise solutions. On SourceForge alone, over 100,000 open source projects are listed, with many more on other open source repositories like CodeHaus, Tigris, Java.net, ObjectWeb, and OpenSymphony (BRR, 2005). Any unexpected or even incorrect functionalities in that software may severely affect users' systems and knowledge base, causing quality and/or security violations.

In our discussions, an object is considered to be of high quality if it fits its intended uses in scientific computing, decision making, and planning (Kahn, *et al.* 2002). Various desirable attributes have been defined for object quality for different usage scenarios, including accuracy, correctness, timeliness, completeness, and relevance (Wang, *et al.* 1993). For a passive entity, such as a document or a file, object quality refers to its correctness and accuracy in describing "things," such as a computational algorithm, a law of physics, scientific data, or some other forms of tangible and intangible knowledge items. For an executable program (e.g., a software package or module), object quality refers to its computational correctness. An active program has high quality if it produces the correct results in a consistent and predictable manner given all reasonable inputs. If a program has been developed with race condition, deadlock, or any logical errors, for example, then that program has an unacceptable level of quality. An executable program's quality also includes a dimension of security; ensuring that a program is reliable and safe to execute is important to guarantee the program's usability. Since object quality is so domain dependent and user-specific, we use the term *trust* to represent how trustworthy an object is in terms of its quality and/or security in an intended usage scenario. In other words, a trust value of an object represents a user's confidence towards the quality and security of the object, e.g., the correctness of the object to reflect a real world situation and the safety and security features of the object if it represents an executable program. Trust is such an important factor for a VO participant to evaluate and choose an object before it can be used.

As we will discuss in more detail, we distinguish two types of trust: (1) subject trust - trust put on a subject, such as a software developer or a data producer, based on the expected behaviors of the subject; and (2) object trust - trust put on an object, such as a software program or an intelligent knowledge item, based on the inherent features of the object and the authenticity of the item. The former is subjective and forward-looking. For instance, a consumer trusts a merchant and expects the merchant to behave honestly in conducting online transactions in the future. The latter is more objective and current-looking. It represents the existing state of the object - is the object of high quality, is it bug-free (for software), or does it accurately reflect a world reality?

Unlike the dominating stream of research work which focuses on subject trust management, the framework presented in this paper studies the object aspect of computational trust as applied in a VO. As we discussed earlier, a VO contains participants with different levels of expertise and information processing cultures. Object quality assurance is challenging in such an environment. We define *object trust management* as "a set of systematic models and approaches to assist an evaluator to assess the trustworthiness of an object in terms of its quality and security based on their intrinsic and extrinsic features. Trust principles, such as trust combination and composition, provide guidelines for object trust

evaluation and selection." As explained in (Zuo & Panda, 2008), an intrinsic feature of an object refers to the inherent property and characteristic of an object. Such a feature is the essential attribute of the object. For instance, the robustness of a software program describes one fundamental functional feature about the program. This feature helps an evaluator in assessing the quality of the software. Hence, this attribute is considered an inherent feature of the software. On the other hand, an extrinsic feature of an object refers to some external factor that is not necessarily part of the object itself but closely related to the object. For example, the reputation and trustworthiness of developers is the extrinsic features of a software object. Other extrinsic features of the software object may include the total number of books published about the software package, the difficulty level to enter the core development team of the software, and the recommendations from adaptive users about the software. As we can see, those extrinsic features can also help a user in evaluating the trust of the object.

We should note that subject and object are two aspects of trust management and are complementary to (but do not replace) each other. However, object trust management is a largely under-studied area. More or less, the current practice is to use subject trust to evaluate object trustworthiness. For instance, how much we trust software largely depends on the trust in and reputation of the software developer. However, relying on subject trust does not always provide a complete and reliable solution to object trust evaluation. As we know, the most experienced software developers also sometimes produce buggy software; it is unavoidable given the large scope and growing complexity of modern software. Focusing on the quality metrics of the object itself (instead of the trust and reputation of its producer) gives a user a level of higher confidence towards the object. Therefore, it is advantageous to assess the intrinsic features of an object and perform trust evaluation at object level (hence the notion of *object trust management* as defined in this paper).

The goal of this work is to specify the principles and models to evaluate the trustworthiness of an object. The evaluation is based on how much the object should be trusted with respect to two perspectives: (1) whether the object is correct and accurate in expressing a topic or issue, and (2) whether the object is secure and safe to execute (in the case of an executable program). Our research methodology includes both qualitative and quantitative analysis approaches. On the qualitative side, it includes a wide-scope literature review, formation of the problem of study, definition of key concepts and variables, specification of object trust principles, and provision of case studies to illustrate the implication of the research in knowledge management. On the quantitative side, we developed formal models, approaches, and algorithms to quantify and implement the proposed concepts and ideas. Together, the studies make the proposed trust evaluation framework theoretically sound and practically significant. More specifically, the research carried out in this paper includes (1) a formal specification of the aspect of trust management – object trust management; (2) two object trust principles as the foundation for trust evaluation; (3) formal approaches, algorithms and models to apply the principles to evaluate the trustworthiness of objects in an VO environment; and (4) two examples to illustrate the proposed concepts and approaches of the trust principles. To the best of our knowledge, this work is the first of its kind to formally define object trust management and study object trust evaluation. It is also the first to study object trust principles and use a component-based approach to evaluate the quality and security of an executable program. The implication of this research is that it provides theoretical analysis and empirical insights about object trust evaluation which is essential in knowledge identification, creation, representation, distribution, and utilization.

The remainder of the paper is organized as follows. We first review related work on knowledge and trust management (Section 2). Then the terminology used in our research is introduced (Section 3). Next, the trust principles and the formal models to implement them (including approaches, algorithms, and illustrative examples) are discussed in more detail (Section 4). Finally, we conclude our paper and also discuss the implication of the research on knowledge management, its limitation, and future work (Section 5).

## 2. LITERATURE REVIEW

Knowledge management (KM) has been a continuous research interest in academia and industry. Akram *et al.* (2011) define KM as an organizational process that aims to create centralized knowledge source within the organization that acquire, assimilate, distribute, integrate, share, retrieve and reuse the internal and external, explicit and tacit to bring innovation in the forms of the product, people and organizational process. Sarrafzadeh, *et al.* (2006) explains KM as the creation and subsequent management of an environment which encourages knowledge to be created, shared, learned, enhanced, and organized for the benefit of the organizations and its customers. Canfora, *et al.* (2002) refer KM as the methods and tools for capturing, storing, organizing, and making accessible the knowledge and expertise within and across communities. Bouthillier & Shearer (2002) proposed a conceptual framework for KM process, which includes discovery, acquisition, or creation of knowledge, storage and organization of knowledge, sharing of knowledge, and use and application of knowledge. Sherif, *et al.* (2012) developed and tested a theoretical framework that examines the capacity of electronic open networks and closed interpersonal networks in building social capital and creating new knowledge.

All the above works highlight knowledge acquisition, creation, sharing, and utilization as the essential components of a knowledge management framework. As the importance of managing knowledge continues to grow and the capacity to generate, gather, model, and retrieve more complex, multi-disciplinary knowledge continuously increases, there is a pressing need for techniques and approaches to ensure the quality of the knowledge elements that are acquired, created, and adopted by organizations. Our research in this paper presents an object trust management framework (including methodology, principles and models) to facilitate knowledge management (e.g., formal evaluation of the accuracy and security of knowledge elements in the processes of knowledge identification, creation, distribution, and adoption) in a virtual organization environment. Since trust is the central theme of this research, we next briefly present some relevant research work on trust management.

The Oxford English Dictionary defines trust as “the firm belief in the reliability or truth or strength of an entity.” Webster’s Dictionary defines trust as “a confident dependence on the character, ability, strength or truth of someone or something.” Both definitions emphasize the subjective nature of trust between two subjects (i.e., subject trust). As we mentioned earlier, our framework focuses on object trust, i.e., the trust that a subject places on an object in terms of its intrinsic features. Like subject trust, object trust is measurable and computable. Blaze (Blaze, *et al.* 1996) and Marsh (Marsh, 1994) are among the first to formalize trust in computational models.

Trust has been a subject of continuous interest in different areas including E-commerce (Piao and Gan, 2009; Kim, *et al.* 2008), online communities (Kim and Phalak, 2012; Zhang, *et al.* 2010), cyber security (Gligor and Wing, 2011; Ma, *et al.* 2009; Squicciarini, *et al.* 2007; Squicciarini, *et al.* 2006), sociology (Ward and Meyer, 2009; Meyer, *et al.* 2008; Mollering, 2001), behavioral and psychological studies (Kang, 2011; Jason, *et al.* 2007), risk management (Earle, 2010; Alcalde, 2009; Lacohee, *et al.* 2006), organizational management (He, at al. 2009; Ronald and Lawrence, 2007), and knowledge sharing (Wickramasinghe and Widyaratne 2012). Various trust models have been developed in a wide range of applications. However, most of those models are conceived in the scope of subject trust management. We briefly mentioned a few here.

In Blaze, *et al.* (1996), subject trust management is noted as a unified approach to specifying and interpreting security policies, credentials, and relationships which allow direct authorization of security-critical actions in distributed systems. It supports delegation and policy specification and refinement at the different layers of a policy hierarchy, thus solving to a large degree the consistency and scalability problems inherent in traditional access control lists. In Josang, *et al.* (2000), trust management is defined as the activity of collecting, codifying, analyzing, and presenting security relevant evidence with the purpose of making assessments and decisions regarding E-Commerce transactions. In Khare, *et al.* (1998), subject trust management is regarded as a new philosophy for codifying, analyzing, and managing

decisions about trust, with regards to the overarching question of "is someone trusted to take action?" According to Chu, *et al.* (1997), trust management provides a systematic means for deciding whether a requested action, supported by credentials, conforms to a specific policy.

A model (Lim, *et al.* 2006) developed on how trust-building strategies increase and affect trust among users investigates two trust-building strategies among different subjects: portal association (based on reputation categorization and trust transference) and satisfied customer endorsements (based on unit grouping, reputation categorization, and trust transference). Research work (Liu, *et al.* 2008) also addresses the problem of predicting whether a user trusts another. A classification approach is proposed to solve the trust prediction problem and taxonomy is developed to obtain an extensive set of relevant features derived from user attributes and user interactions in an online community. Empirical results show that the trust among users can be effectively predicted using pre-trained classifiers. Studies by Pavlou, *et al.* (2004) are conducted about the sociological and economic theories applied to institution-based trust. The authors propose that the perceived effectiveness of three IT-enabled institutional mechanisms (feedback mechanisms, third-party escrow services, and credit card guarantees) engender buyer trust in online auction sellers. In Turel, *et al.* (2008), the authors study how justice and trust affect user acceptance of e-customer services by conducting an online experiment involving 380 participants. The results suggest that trust in e-customer service fully mediates the effects of trust in the service representative and procedural justice on intentions to reuse the e-customer service.

Unlike all the above research, in this paper, we study object trust, a different aspect of trust management. We discuss the trust models and principles for a subject to assess the trustworthiness of an object in terms of its quality and/or security features. A short note by Michael *et al.* (2001) highlights the importance and necessity of object trust: intelligent agents can check certificates to validate sites, but how can they determine the accuracy of the information located at the sites? When downloading information from a web site, how does the agent know whether the information contains malicious code until it is too late to prevent the malicious code from executing? In a VO, analyzing an object's intrinsic features is important since it helps a user make a decision regarding whether the object is trustworthy (and hence can be used). Currently the decision to accept and use any external information is largely made in an ad hoc fashion. Focusing on the study of information quality and security at the object level gives a user a higher level of confidence to assess and select secure and high-quality objects. Unlike subject trust management, object trust management specifies guidelines to analyze the object itself (not its owner or producer), focusing on studying the characteristics (both intrinsic and extrinsic) of the object.

## 3. TERMINOLOGY

### *3.1 The Object and Its Component Graph*

As we mentioned earlier, an object represents an intellectual item such as a software program, a database item, a document file, or other forms of knowledge items. An object has a set of values, called *attribute values*, representing the inherent features of the object. For instance, a database tuple about an employee record has a set of attribute values such as the employee's name, ID number, rank, responsibility, etc. The attribute values of a software program refer to its designed functionalities or specifications. Each object has a central theme and is about some *topic*. Different objects in the same category may have different attribute values on the topic, just as different people may have different opinions towards a particular issue.

From an information-processing perspective, two types of objects can be specified. A *simple object* is an atomic intellectual entity, e.g., a single software module, or a basic database item. Such an object does not need any component and is considered "indivisible." A *compound object*, on the other hand, is a composite entity constructed from a set of component objects through a composition process. The composition process, also known as the computational procedure, is expressed by a set of composition functions. Object composition has been used intensively in information-processing and software

development. Component-based software engineering, for instance, has been widely used to modularly build large-scale and complex software systems from existing fully-tested modules. The goals, among others, are to consistently increase return on investment and reduce the time to market, while assuring higher quality and reliability than can be achieved through component reuse and integration-based software development. The use of off-the-shelf or open source software components is becoming an economic and strategic necessity for organizations in a wide variety of application areas including finance, medicine, logistics, administration, manufacturing, and commerce.

A piece of integrated software is considered a compound object in our discussion. The individual software components, however, can be simple objects or compound objects themselves (if they are further based on other software components). In general, if an object is composed of other objects, it is said to be dependent on those objects. Object dependency is transitive, i.e., object $O_k$ is indirectly dependent on object $O_t$, if some intermediate object $O_i$ exists such that $O_k$ depends on $O_i$ and $O_i$ depends on $O_t$. In this case, $O_t$ is called a sub-component of $O_k$.

A chain of object dependency therefore represents the components and sub-components of a compound object. Searching the component information of an object generates a directed graph called a *component graph*, for that object. A component graph represents a logical view of how the components and sub-components have been integrated to form the compound object. The formal definition of a component graph is given below.

**Definition**: The *component graph* of object $O$ is a directed graph $G = \{V, E\}$, where $V$ represents a set of vertices (or nodes) and $E$ represents a set of edges. $G$ has the following characteristics:

(1) A node $v \in V$ is either a *formula node* or an *object node*. A formula node, denoted as a rectangle, represents the composition functions used to form the compound object. An object node, denoted as an ellipse, represents an object, or one of its components or sub-components;

(2) For a directed edge $e = (v, v') \in E$, if $v$ is an object node, then $v'$ must be a formula node. This indicates that the composition functions represented by $v'$ are used to construct the object represented by $v$;

(3) For a directed edge $e = (v, v') \in E$, if $v$ is a formula node, then $v'$ must be an object node. This indicates that the object represented by $v'$ is used as a component to construct a compound object;

(4) Since object $O$ itself has only outgoing edges but no incoming edges, it is defined as the special "root" of $G$. Similarly, a simple object in $G$ has only incoming edges and no outgoing edge and therefore is called a "leaf" of $G$;

(5) Formula nodes and object nodes appear alternately along any path from the root to a leaf. The *depth of G* is defined as the number of formula nodes (or the number of object nodes, minus the root), along the longest path from the root to a leaf.

**Algorithm 1**: Component Graph Construction for Object *O*

```
void Construction(O) {
  1. V = ∅;   E = ∅;
     // The vertex and edge sets are initially set to empty.
  2. V = V ∪ {O};
  3. Let cfs represent the compositing functions to construct O;
  4. V = V ∪ {cfs};      E = E ∪ {(O, cfs)};
  5. For every component c of object O, do
        5.1 E = E ∪ {(cfs, c)};
        5. 2 V = V ∪ {c};
        5.3 If (CompoundObject(c)), do
              5.3.1 Construction(c);
  6. Return G = {V, E} as the constructed component graph for O;
}
```

**Figure 1**. Algorithm to Construct the Component Graph of Object *O*

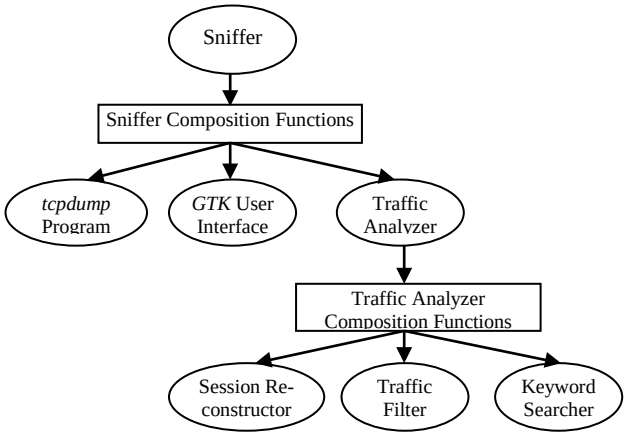


**Figure 2**. Component Graph of the *Sniffer* Program

Building the component graph of an object requires identifying the components and the composition functions that are used to construct the object. Since we study object trust in a VO environment, which is different from a completely open system such as the Internet, we assume that rules or conventions have been placed to require that the component information of an object is available. Consider again the case of the open source software community. The source code and software architecture are available to other members (and even to the public) for further development. In this case, the component information can effectively be kept track of. In addition, the necessary software engineering documents such as code comments and technical files describing how the software was developed are typically provided by the software developers. The providers are motivated to do so since the availability of this information would encourage more users to select and use their software modules. This makes trust evaluation based on component information possible as we will discuss later. Next, we show a systematic way to construct a component graph from the component information that is given about an object.

An algorithm to construct a component graph for a compound object *O* is given in Figure 1, where *V* represents the vertex set and *E* represents the edge set of the component graph to be constructed. *V* and *E* are initially defined as empty. Basically, the algorithm identifies the components and sub-components of the object *O* and builds the component graph {*V*, *E*} recursively. The function *CompoundObject*(*c*) evaluates whether an object *c* is a compound object. If so, the function recursively calls itself with the component *c* as the argument.

### *3.2 An Example of a Component Graph*

In this section, we present an example component graph of a software program that is the result of an open source software development project (see Figure 2). Suppose that a network management software package *Sniffer*, is under development, which is to be used by system administrators to monitor and analyze network traffic with necessary user interfaces. Since several existing functional modules can be re-used as software components, the *Sniffer* software can incorporate those functions. For instance, the well-known software modules such as *tcpdump* and *Windump* are available as downloadable programs that provide functions to intercept and monitor network packets using a command-line mode. Those functions are essential for the *Sniffer* program. Therefore, the developers may decide to incorporate that existing software as components into the *Sniffer* software. To provide complementary user interfaces, a software library, *GIMP* Toolkit (*GTK*), is available and can be re-used in building *Sniffer*. Furthermore, to provide solid data analysis, the proposed *Sniffer* program will be designed with a traffic analyzer program, which can be constructed based on three existing software programs: a traffic filter program (*TF*), a session reconstruction program (*SR*), and a keyword searching program (*KS*). *TF* provides a user with the ability to specify the types and attributes of network traffic to be monitored and recorded. Those functions are crucial since they can limit the huge amount of network traffic data to a manageable size with only those of interest. The specification rules also include such attributes as network protocols, source and destination IP numbers, and sending and receiving timestamps. *SR* combines network traffic data and then reconstructs them into communication sessions between the senders and receivers. *KS* allows a user to specify keywords for search in order to identify their data of interest. Based on the above information, the component graph of *Sniffer* has been constructed as shown in Figure 2 by applying Algorithm 1 (due to page limitations, we will not give the details of the component software composition process).

## 4. OBJECT TRUST PRINCIPLES

Object trust principles specify the rationale and guidelines to evaluate the trustworthiness of an object in terms of its quality and/or security features. Unlike subject trust, object trust is neither transitive nor symmetric. But object trust is “composible” and “combinable” as we will discuss these two properties in detail in the sections that follow. As in many trust models found in literature, the trust of an object in our

framework is expressed as a numerical value in [0, 1], 0 being untrustworthy and 1 being very trustworthy.

The two principles discussed in this section describe object trust from different aspects. Hence, they are complementary and applied to different cases. For instance, are multiple objects with the same topic available (for trust combination)? Can the trustworthiness of the components and the composition functions of an object be readily assessed (for trust composition)? However, the two principles can be used in tandem to better evaluate the trustworthiness of an object. We next discuss the two principles in detail.

### *4.1 The Principle of Trust Composition*

Much data in scientific and business computations is composite, i.e., the data is formed from a set of components through some composition processes. Just as in component-based software development, a software program is assembled from the components modules and these components in turn may be constructed from their components as well.

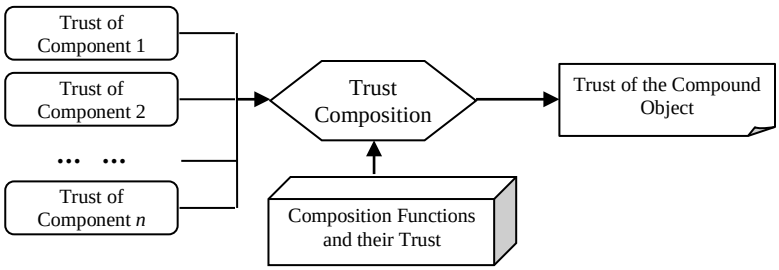


**Figure 3**. Logical View of Trust Composition Process

Object composition can facilitate the analysis of the trustworthiness of an object, since the trustworthiness of an object depends largely on how its components can be trusted. In addition, the way that the components are integrated to form the object also affects the trustworthiness of the object. People are rightfully skeptical of data based on unsound individual pieces and questionable formatting processes. In software development, low quality of source objects implies low quality of any object based on them. Mann (2012) supports the idea of recording the method used to form integrated data in such a way that a user can readily assess whether the data can be trusted for a particular analysis. If a user has a way to study the components of an object along with the detail about how the object has been formed, the user would be able to more accurately evaluate the trustworthiness of the object. Tracing the components of an object would give the user insight into the intrinsic features of the object and enable the user to build more confidence in the quality and security features of the object.

#### 4.1.1 The Principle

The principle of trust composition specifies guidelines to calculate the trust value of a compound object formed from a set of components. A logical view of trust composition is illustrated in Figure 3, where the trust values of both the components and the composition functions are composed – leading to the final trust value of the compound object. The principle is specified below.

> ***Principle of Trust Composition***: the trustworthiness of a compound object can be assessed based on the trustworthiness of its components and the composition functions used to form the object. The trust values of the components are "composable" to get the trust value of the compound object.

#### 4.1.2 A Component-based Model for Object Trust Composition

To apply the principle of object trust composition, two things must be determined: (1) how the object is formed, i.e., the way the object is composed from its components; and (2) how those components can be

trusted. The first issue is to determine whether the composition functions are correct and appropriate, thus trustworthy. The second one is to verify whether the components are free of errors and/or secure to use. The trust value of the object depends on both of these factors.

Similar to the transitivity rule in subject trust calculation, the trust value of a compound object can be calculated as the product of the trust values of its components and those of the composition functions, as shown in Formula (1), where $T_O$ represents the trust value of a compound object $O$; $T_{cps}$ represents the trust value of the components of $O$, called *overall component trust*; and $T_{cfs}$ represents the trust value of the composition functions in terms of the correctness and appropriateness of the process used to integrate the components to form $O$.

$$T_O = T_{cps} * T_{cfs} \qquad (1)$$

In the following sub-sections, we discuss technical approaches to calculate the parameters used in Formula (1). We also give an algorithm to formally represent the trust computation process.

#### *4.1.2.1 Overall Component Trust $T_{cps}$*

The first step in evaluating the trustworthiness of a compound object is to verify its components. $T_{cps}$ represents the overall trust of all the components of $O$ in terms of their quality and security features. Let the set $\{c_1, c_2, \ldots, c_n\}$ represent the components of $O$. Let $T_{Ci}$ represent the trust value of the $i^{th}$ $(0 < i \leq n)$ component of $O$, i..e, $c_i$. According to Biba's model (Biba, 1975), a subject that reads an object with a lower integrity will have its own integrity decreased. The model implies that the low integrity of a source object indicates low integrity of any object constructed from that same source object. A similar idea is also represented in the principle of minimum specificity (Dubois and Prade, 1987) for reasoning uncertain information. Hence, a set of objects cannot have an overall trust value higher than that of any of its components. Therefore, $T_{cps}$ can be calculated as the minimum of all the trust values of the components $T_{C1}, T_{C2}, \ldots, T_{Cn}$ as shown in formula (2).

$$T_{cps} = Min(T_{C1,} \ldots, T_{Ci, \ldots,} T_{Cn}) \qquad (2)$$

#### *4.1.2.2 Composition Functions Trust $T_{cfs}$*

The trust value of a set of composition functions represents the confidence that an evaluator places on the correctness and appropriateness of the way that the components are integrated to construct an object. Based on the BRR model (BRR, 2005), we propose four steps to evaluate the trustworthiness of the composition functions used to form a compound object. We intentionally leave our approach general so that users can set their own assessment criteria, apply their own domain knowledge, and perform their own specific evaluation.

*Step 1*: Based on the usage requirements of the object, specify a set of metrics, say $\{m_1, m_2, \ldots, m_n\}$, which are used to assess the quality properties of the composition functions. The goal is to obtain a set of reproducible, objective, and quantifiable measurements, which highlight the prominent characteristics of the composition functions and the object under evaluation. Each metric can be a quantitative or qualitative variable.

*Step 2*: Rank $\{m_1, m_2, \ldots, m_n\}$ according to the importance in the specific usage context of the object and assign the corresponding weight $w_i$ to each $m_i$, i.e., $\sum_{i=1}^{i=n} w_i = 1$. Each weight represents how much a metric contributes to the final evaluation result of the composition functions.

*Step 3*: Gather assessment data for each metric $m_i$ in terms of the properties of the composition functions. Transform the assessment result to a normalized score $s_i \in [0, 1]$.

*Step 4*: Calculate the weighted average $\sum_{i=1}^{i=n} (w_i * s_i)$, which is the trust value $T_{cfs}$ assigned to the composition functions under evaluation.

**Table 1**: Workflow of Calculating the Trust Value of Composition Functions

| Assessment Metrics | Explanations | Measurements (metric values) | Noramlized scores | Metric weights | Averages |
|---|---|---|---|---|---|
| Functionality ($m_1$) | The composition functions are correct (e.g., the procedures are logcally sound and the algorithms are well justified) and closely follow their pre-designated specifications | Very good | 0.9 | 25% | 0.225 |
| Soundness of software architecture ($m_2$) | There is a well-defined, flexible and modular software architecture to integrate and assemble the components to complete software | Excellent | 1 | 20% | 0.2 |
| Component cohesion and interoperability ($m_3$) | The components have been correctly integrated – components are compatible and interoperable with each other through well-defined interfaces. Potential conflicts have been solved (e.g., through proxy or function wrapping provided by the compostion functions) | Acceptable | 0.6 | 15% | 0.09 |
| Component inter-communications ($m_4$) | There are well-defined channels for inter-process and inter-application communications, including methods for message passing, synchronization, shared memory, and remote procedure calls | Good | 0.8 | 15% | 0.12 |
| Level of testing activities ($m_5$) | There is a completion of system and integration testing for the composition functions following well-defined testing strategies. Reports on benchmarks are available | Fair | 0.3 | 15% | 0.045 |
| Soundness of software engineering ($m_6$) | Composition functions are specified following sound software engineering processes | Very good | 0.9 | 10% | 0.09 |
| Documentation ($m_7$) | There exist various software documentations including development, functionality, configuration, optimization, and user documents | Good | 0.8 | 5% | 0.04 |
| **Total** | | | | 100% | **0.81** |

Next, we use the example of the *Sniffer* program as given in Figure 2 to illustrate the above approach. For the sake of simplicity, we limit our discussions to only calculating the trust value of "*Sniffer* Composition Functions." As shown in the figure, the set of composition functions logically link the three components "*tcpdump* program," "*GTK User Interface*," and "*Network Traffic Analyzer*" to form the complete software "*Sniffer*."

Our evaluation process is summarized in Table 1. Following the sound software engineering practice and standard feature-set required for an average use of the *Sniffer* software, seven representative assessment metrics have been identified (as shown in the first column of Table 1). Their explanations are also given in the second column of the table. The measurement for each metric can be determined by a variety of approaches (depending on how rigious the evaluation would be), including (1) expert inputs; (2) formal analysis, such as automatic code review, dynamic behavior anaysis, and structural and data flow analysis; and (3) model-based reasoning, such as Bayesian reasoning, model checking, and formal verification. Several tools are also avaialble for automatic or semi-aotomatic software verification, such as finite state machines, labelled transition systems, Petri-nets, timed automata, process algebra, and formal sematics of programming languages. In addition, techniques such as proof-carrying code and theory proof exist to provide machine-checkable code to verify the safety and functionality of a software program. Based on those approaches, the measurement for each metric is obtained and transformed to a normalized scale in the range of [0, 1]. They are shown in the third and fourth columns in Table 1, respectively.

To calculate the weighted average of the trust value of the compositon functions, a systematic way to determine the metric weights is required, i.e., how much each metric contributes to the final result.

Relative ranking or weight factors among a set of metrics based on their significance/importance related to intended usage of the object can be determined using the "zig-zag" method as proposed in the BRR mode (BRR, 2005). We assume that based on expert inputs, the seven metrics $\{m_1, m_2, \ldots, m_7\}$ in Table 1 have already been ordered according to their decreasing importance. The following two steps will make a full-scale assignment of weight of each metric: (1) assign the average weight of 15% to the median ranking, which is $m_4$ in our example; and (2) work in a "zig-zag" pattern to assign weight to each of the remaining metrics. The metrics higher on the list ($m_1$, $m_2$, and $m_3$) would be assigned weight of at least 15% and the metrics lower on the list ($m_5$, $m_6$, $m_7$) would be assigned weight no higher than 15%. In our example, we set the weight for the most important metric as 25%, i.e., $w_1 = 25\%$ and the weight for the least important metric as 5%, i.e., $w_7 = 5\%$. These and the remaining weights are represented in Table 1. It is clear that the sum of all the weights is 100%.

Finally, the trust value of "*Sniffer* Composition Functions" is calculated by the weighted average of the scores on these assessment metrics. As we can see in Table 1, the trust value of the composition functions is calcualted as 0.81.

#### *4.1.2.3 The Algorithm*

We can now apply Formula (2) to Formula (1) and consequently obtain Formula (3) as shown below.

$$T_O = T_{cps} * T_{cfs} = Min(T_{C1}, \ldots, T_{Ci}, \ldots, T_{Cn}) * T_{cfs} \qquad (3)$$

To use Formula (3), the trust value of each individual component must be determined. As we mentioned earlier, a component could be a compound object itself. Therefore, to calculate the trust value of a compound component, we can apply Formula (3) recursively to each of its components. This "divide and conquer" style process continues until a base case is available. While a complex compound object is hard to evaluate (trustworthy), a base component with a smaller scope and less complexity is much easier to investigate and verify (we call such an object a *basic component*). Based on this idea, we have developed an algorithm (see Algorithm 2 in Figure 4) to calculate the trust value of a compound object *O* recursively based on its components and the composition functions used to form *O*. In the algorithm, function *trustValueAvailable*(*O*) returns *true* if a base case to calculate an object's trust value is available, or *false*, otherwise; $TB_O$ represents the trust value of object *O* in such a base case (as will be discussed next); and $T_{cfs}$ represents the trust value of the composition functions of *O* as we discussed in the previous section.

Since the base case to calculate the trust value of an object is essential to apply Algorithm 2, we discuss two base cases where the trust value of a basic software component is readily available or can be calculated.

***Base Case One:*** *subroutines or certified basic components* This type of base case occurs when a basic software component is reduced to a subroutine library or sub-function, which has been fully documented, thoroughly tested and developed based on reused, well-defined algorithms and design patterns in an effective manner. If that is the case, the basic component is considered with a high trust value. There is a broad continuum in the quality of open source or on-the-shelf software. Widely adopted source packages, backed by a foundation, a corporation, or a strong community, often have high quality. Since a basic component is much less complex and has a much smaller scope as compared to a compound object, it is relatively easier to be tested and verified. If such a basic component is certified by a well-known, reputable organization, it can be accepted by a user with a high trust value. Furthermore, industry performance testing and benchmark reports often list some well-known, fully-tested software components, which have the minimum number of bugs and deficiencies. We consider each of those basic software components as a base case since they have been fully tested or authorized by the industry leading organizations. In a VO, there may be a short list of such "elite" basic software components with a

high level of reputation of being "good" objects in the community. Their trust values are considered high and therefore known to each participant of the VO.

**Algorithm 2**: Calculation of Trust Value of Object *O*

*real_number trustCalculation*(*O*) {
1. If(*trustValueAvailable*(*O*)), do
   1.1 Return $TB_O$;
2. Else, do
   2.1 For every component $c_1, c_2, \ldots c_n$ of *O*, do
      2.1.1 Let $Tc_i$ represent the trust value of $c_i$ ($1 \le i \le n$);
      2.1.2 $Tc_i = trustCalculation(c_i)$;
3. Return $Min(Tc_1, Tc_2, .., Tc_n) * T_{cfs}$;
}

**Figure 4**. Algorithm to Calculate the Trust Value of Object O

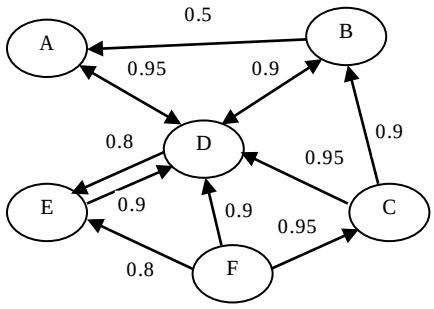


**Figure 5**. Example of Web of Trust

***Base Case Two:*** *component recommendation* Another base case occurs when an evaluator consults the trustworthiness of a component object from a third party, called a recommender, who can assess the quality and security features of a software component based on his/her expertise. As we mentioned earlier, subject trust and object trust are complementary and the former can assist the latter when appropriate. More specifically, the trust value of a basic component, say *c*, i.e., $TB_c$, can be calculated as the product of the trust value of the recommender and the trust value of the object as ranked by the recommender. Discounting the recommended value by the trustworthiness of the recommender itself is necessary because the recommender may not be fully trusted in making a qualified recommendation in terms of the object of concern. Furthermore, a recommender may only provide this trust value based on his/her limited knowledge. To take this or other types of inaccuracy into consideration, we calculate the average of *n* recommendations from different entities about the same object as shown in Formula (4), where $T_{Ri}$ represents the trustworthiness of a recommender $R_i$ ($1 \le i \le n$) for an valuator (i.e., the user to evaluate the trust value of an object); and $T_{Ri,c}$ represents the trust value of component *c* as recommended by $R_i$.

$$TB_c = \sum_{i=1}^{i=n} (T_{Ri} * T_{Ri,c}) \Big/ n \qquad (4)$$

To determine the trust value of a recommender, we propose an efficient approach based on the web of trust as applied in a VO environment. A web of trust describes a direct trust relationship between a pair of neighboring subjects. One subject is a neighbor to another if one of them has built and maintained a first-hand trust relationship with another through some kind of interactions between them (e.g., business transactions, resource sharing, or contractual activities). A subject evaluates the performances of its neighbor after each transaction and uses the evaluation result to dynamically update the trust value for the neighbor. Those direct trusts therefore form a web of trust, which represents a direct trust relationship between each pair of subjects in a VO.

Formally, a web of trust is represented as a weighted directed graph, where each vertex represents a subject and each directed edge represents the trust orientation and a trust degree. For instance, an edge with a weight *w* ($w \in [0, 1]$) from a vertex *V* to another vertex *V*' represents that the subject denoted by *V* directly trusts the subject denoted by *V*' with a degree *w*. Figure 5 gives an example of a web of trust with six subjects.

Although a web of trust represents direct trust relationships between subjects, indirect trust between any pair of non-adjacent subjects can be calculated based on the principle of trust transitivity. Since there could be multiple paths between a pair of subjects, we use the maximum principle to combine trusts from

different paths of a trust network – a user believes anything that is believed by at least one of the users she trusts. Based on this principle, indirect trust that an evaluator (trustor), say $V_1$, puts on a recommender $R_i$ (trustee), can be calculated based on trust propagation as shown in Formula (5).

$$T_{Ri} = Max(\forall P = (V_1 \rightarrow V_2 \rightarrow \ldots \rightarrow V_m \rightarrow R_i): (\prod_{k=1}^{k=m-1} T_{V_k\_V_{k+1}}) * T_{Vm\_Ri}) \quad (5)$$

In the above equation, $P$ represents a path of neighboring nodes ($V_1$, $V_2$, …, $V_m$, $R_i$) from $V_1$ to $R_i$ in a web of trust. We use the symbol $T_{Vk\_Vk+1}$ to represent the direct trust that a node $V_k$ ($1 \leq k \leq m$-1) puts on its direct neighbor $V_{k+1}$ along the path $P$. As we discussed above, this direct trust value is represented by the weight of edge ($V_k$, $V_{k+1}$) in the web of trust. Similarly, $T_{Vm\_Ri}$ is used to represent the trust that the second last node $V_m$ in the path $P$ puts on the recommender $R_i$. Function $Max(.)$ indicates that $T_{Ri}$ is the maximum trust rating resulted from all these paths given the evaluator $V_1$ and the recommender $R_i$.

By applying Formula 5, an indirect trust value between a pair of subjects can be calculated. For instance, the trust value of subject $A$ (trustee) for subject $E$ (trustor) as shown in Figure 5, denoted as $T_A$, is calculated based on the path $P = (E \rightarrow D \rightarrow A)$ as shown below (since this path results in the maximum trust value of $A$):

$$T_A = T_{E\_D} * T_{D\_A} = 0.9 * 0.95 = 0.855$$

After the trustworthiness of a recommender is determined, the trust of a basic component can be calculated using Formula (4). As we discussed earlier, determining this base case is an essential step in calculating the trust value of a compound object as shown in Algorithm 2.

### *4.2 The Principle of Trust Combination*

Trust combination is applied in a case when at least two objects with the same topic in consideration are available. Different from the principle of trust composition, which is built based on the trustworthiness of the components and the composition functions to form an object, the principle of trust combination is applied to two or more objects with significantly dissimilar component structures but having similar attribute values on the same topic. We first use an example to illustrate the idea and then present a formal model to apply the principle.

#### 4.2.1 A Motivating Example

As we mentioned earlier, a set of objects may exist on a topic and be constructed by different subjects (i.e., VO participants). But the attribute values of those objects could be very different. It is easy to understand that those subjects may have used different approaches and data in generating those objects. Given different attribute values of those objects, a user may be uncertain which object is more trustworthy. However, if the user compares the attribute values of those objects (on the same topic) and the way they are formed, the new finding may improve the user's confidence towards the trustworthiness of those objects. For instance, by comparing the attribute values and the component structures of two objects, the user may identify some "multiple-proof" patterns as evidence to show the correctness of the two objects. Then the trust of the two objects can be evaluated to a higher value. More specifically, if those two objects share similar attribute values that are calculated through significantly different methods or obtained independently by different subjects, then the objects would be more trustworthy than if they were evaluated separately.

We use an example from Zuo, *et al.* (2006) to illustrate the idea of trust combination. Suppose that earthquakes occurred frequently in a remote island in the past few years. A group of scientists are attempting to predict the possibility of earthquake occurring in the next few months. They collect samples of water, dirt, and air on the island. Those samples are then sent to a lab for analysis using scientific tools. After sufficient studies, the group of scientists announces that it is highly possible that an earthquake would occur in the next month. At the same time, another team of scientists is working on the same

project but is using a completely different approach. They observe animal behaviors and other biological and ecological changes on the island. After collecting sufficient evidence, the second team analyzes and summarizes their findings. Then they make a similar announcement that an earthquake would occur during the next month as well. Given the two sources of conclusions about the same issue, we can see that the results are so similar but the conclusions were reached through significantly different approaches. Hence, it is more convincing that an earthquake could occur as compared with the case when only one report is reviewed. Therefore, we say that the trust values of two objects can be combined to a higher value in this case.

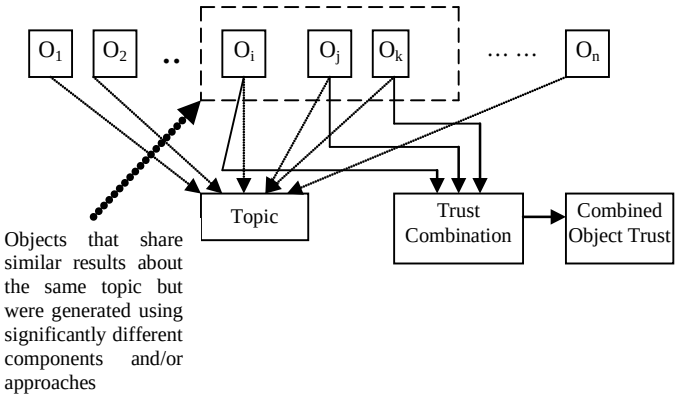


**Figure 6**. Logical View of Trust Combination Process

### 4.2.2 The Trust Combination Principle

The above idea is specified by the principle of trust combination as shown below. To generalize the concept, consider *n* objects, which express different subjects' views on a given topic (as shown in Figure 6). If we can identify some of those objects $\{O_i, O_j, \ldots, O_k\}$ that share similar results but were generated using different components and/or through significantly different approaches, we will be more confident towards the quality of the attribute values expressed by those objects. Consequently, the trust values of those objects can be "combined" to a higher level.

> ***Principle of Trust Combination***: If multiple objects about the same topic express similar attribute values but those objects have been generated by using significantly different components and/or approaches independently, then the trust values of those objects can be combined to form a higher trust towards those similar attribute values.

### 4.2.3 A Trust Combination Model

To apply the trust combination principle, we need to analyze the set of objects (with the same topic) to identify how similar their attribute values are and how differently they are formed. In the following discussions, we illustrate our approach by comparing a pair of objects. But the method can be extended to any number of objects.

Comparing the attribute values of two objects $O_1$ and $O_2$, denoted as $valueSimilarity(O_1,O_2)$, is dependent on the format and the structure of the attribute values of those objects. A common method to compare two sets of attribute values is to use the Euclidean distance as shown in Formula (6) (Zuo & Panda, 2006), where $I_k$ and $J_k$ are the coordinates of $O_1$ and $O_2$ in dimension $k$ (we assume that the attributes in comparison have $N$ dimensions). Jaccard coefficient method (Jaccard Index, 2013) can be used to compare two objects with binary attribute values. Measuring the similarity between two text documents can be conducted by employing clustering techniques, which essentially group similar documents into the same set.

$$valueSimilarity(O_1,O_2)= \sqrt{\sum_{k=1}^{k=N}(I_k - J_k)^2} \quad (6)$$

Given the specific component structure of an object, we next discuss the techniques which are particularly applicable to our framework to identify how differently two objects are constructed (e.g., using significantly different components and approaches). As defined earlier, a component graph is a data structure representing how an object is formed and with which components. Hence, we can translate the task of calculating the dissimilarity of two objects to comparing the dissimilarity of their corresponding object component graphs.

Based on Zuo, *et al.* (2006 & 2007), an algorithm (see Algorithm 3 in Figure 7) is presented to calculate the degree of dissimilarity between two objects. The main idea is to quantify the structural difference between the component graphs of the two objects using an error tolerant graph/sub-graph isomorphism approach. This approach is based on graph edit distance, i.e., compensating the distortions in the input graph by means of a series of edit operations. Intuitively, we want to measure the least cost to transform the two component graphs in such a way that the transformed graphs are isomorphic. There are two main functions in the algorithm: (1) *weightAssign*(*G, w*) and (2) *graphTransformCost*($G_1$, $G_2$). Those two functions are discussed in more detail next.

To measure the edit cost to transform the graphs, a weight is assigned to every node of the two graphs (a node represents an object component or a set of composition functions). As shown in Figure 7, the function *weightAssign*(*G, w*) recursively assigns a weight to each node of a graph *G*. The weight of the root object is given as *w* (an arbitrary number). The assignment follows the "proportion and totality" rule: the weight of a compound object is proportionally assigned to its composition functions and all the components. Since the weight of the root has been assigned as *w*, the weights to the composition functions and to the set of components of the root are determined as ($\alpha * w$) and ($\gamma * w$), respectively ($\alpha + \gamma = 1$). If there are *n* components of the root, the weight assigned to each component is ($\gamma * w$) / *n*. The function in Algorithm 3 is called twice in order to assign a weight to each of the nodes of the two component graphs $G_1$ and $G_2$, respectively.

The function *graphTransformCost*(*G, G'*) represents the graph edit and cost accumulation of a source graph *G* toward a target graph *G'*. Graph transformation is to edit the source graph in such a way that after transformation it only contains identical nodes in the target graph. *graphTransformCost*(*G, G'*) calculates the edit cost for such a transform by quantifying each cost to delete a node in the source graph or substitute one node with another. The edit cost is measured based on the degree of similarity and weights of the corresponding nodes of the source and target graphs. Intuitively, the higher is the cost to make a transformation, the more dissimilar those two graphs are; therefore, we use this cost value to measure the dissimilarity of the component structures of the source and target graphs. As we can see in Algorithm 3, the function *graphTransformCost*(.) is called twice with exactly opposite pair of source and target graphs. This is to accumulate all the required edit cost to transform the two graphs so that the transformed graphs are isomorphic. More specially, *graphTransformCost*($G_1$, $G_2$) transforms $G_1$ to some transformed graph $G_t$ (not shown explicitly) so that for each node $v \in G_t$, we have $v \in G_2$. *graphTransformCost*($G_2$, $G_1$) transforms $G_2$ to some transformed graph $G_t'$ (not shown explicitly) so that for each node $v' \in G_t'$, we have $v' \in G_1$. It is easy to see that after the two function calls, the two graphs must be isomorphic.

As we can see, transforming a source graph $G_1$ towards the target graph $G_2$ is conducted concurrently in a layer-by-layer fashion starting from the roots of $G_1$ and $G_2$ (as mentioned earlier, each component graph of an object has a special node with only outgoing edges but no incoming edge. Therefore, we call such a node the root of the graph). On each layer, the algorithm first identifies all the pairs of "comparable" nodes. A pair of nodes are comparable if (1) they are of the same type (either object nodes or composition functions nodes); (2) they are from the two component graphs $G_1$ and $G_2$, respectively; and (3) their parent nodes represent two objects on the same topic. If a pair of such comparable nodes, say

**Algorithm 3:** Calculation of the Component Structure Dissimilarity

**Input**: (1) Compound objects $O_1$ and $O_2$ and their corresponding component graphs $G_1$ and $G_2$ with roots $r_1$ and $r_2$, respectively;
(2) Initial weights $w_1$ and $w_2$ assigned to $r_1$ and $r_2$, denoted as $W(r_1) = w_1$ and $W(r_2) = w_2$.
**Output**: Component structure dissimilarity between $O_1$ and $O_2$, denoted as *structureDissimilarity*($O_1$, $O_2$).

1. For each node $v \in G_1 \cup G_2$ so that $v \neq r_1$ and $v \neq r_2$, do $W(v) = 0$; // Initialize the weight of each node of $G_1$ and $G_2$ to zero
2. *weightAssign*($G_1$, $w_1$); *weightAssign*($G_2$, $w_2$); // Call the function to assign a weight to each node of $G_1$ and $G_2$
3. $TC_{1\_2}$ = *graphTransformCost*($G_1$, $G_2$); $TC_{2\_1}$ = *graphTransformCost*($G_2$, $G_1$);
4. *structureDissimilarity*($O_1$, $O_2$) = $(TC_{1\_2} + TC_{2\_1}) \Big/ \sum_{c_i \in G_1} W(c_i) + \sum_{c_j \in G_2} W(c_j)$

// A function to assign a weight to each node of a graph $G$. The weight of the root of $G$ is given as $w$
*void* ***weightAssign***($G$, $w$) {
1. Let $r$ represent the root of $G$; Let $W(r) = w$;
2. Let $cfs$ represent the formula node of $r$ and $\{c_1, c_2, \ldots, c_n\}$ represent the component nodes of $r$, do
2.1 $W(cfs) = \alpha * w$; // $0 \leq \alpha \leq 1$
2.2 For each component $c_i$ in $\{c_1, c_2, \ldots, c_n\}$, do
2.2.1 $W(c_i) = (\gamma * w) / n$; // $0 \leq \gamma \leq 1$ and $\alpha + \gamma = 1$
2.2.2 Let $G_{cj}$ represent the sub-graph of $G$, which is "rooted" on $c_i$;
2.2.3 *weightAssign*($G_{cj}$, $W(c_i)$); // Recursively assign the weights to the nodes of the sub-graph
}

// A function to calculate the edit cost to transform a graph $G$ to an intermediate graph $G_t$ so that for each $v \in G_t$, we have $v \in G'$
*real_number* ***graphTransformCost***($G$, $G'$) {
1. Let $TC = 0$; // A local variable representing the accumulated edit cost to transform $G$ toward $G'$
2. Let $r$ and $r'$ represent the roots of $G$ and $G'$, respectively;
3. If $r$ and $r'$ are identical, do nothing. // No cost is required to transform $G$ in terms of the root $r$
4. Else if $r$ and $r'$ are not identical but about the same topic, do // $r$ and $r'$ are partially similar
// $TC$ is updated based on the cost of substituting $r$ with $r'$
4.1 $TC = W(r) * (1 - [Attributes(r) \cap Attributes(r')] / [Attributes(r) \cup Attributes(r')])$;
5. Else, do // $r$ and $r'$ are about different topics; therefore, $r$ and $r'$ have no similarity
5.1 $TC = W(r)$; // $TC$ is updated based on the cost of deleting $r$ from $G$
6. If $r'$ represents an atomic object, do // $G'$ is a single-node graph
6.1 For each node $v \in (G - \{r\})$, do // $TC$ is incremented based on the deletions of the nodes left in $G$
6.1.1 $TC = TC$ + *subGraphDeleteCost*($G_v$); // $G_v$ represents the sub-graph of $G$ rooted at $v$
6.2 Return $TC$;
7. Else if $r$ and $r'$ are both compound nodes, do
7.1 Let $cfs$ represent the formula node of $r$ and $L = \{c_1, \ldots, c_n\}$ represent the set of component nodes of $r$;
Let $cfs'$ represent the formula node of $r'$ and $L' = \{c'_1, \ldots, c'_m\}$ represent the set of component nodes of $r'$;
7.2 $TC = TC + \beta_{cfs\_cfs'} * W(cfs)$; // $\beta_{cfs\_cfs'} \in [0, 1]$ represents the functional similarity between $cfs$ and $cfs'$
7.3 For each pair $(c, c')$ satisfying: (1) $c \in L$, and $c' \in L'$; and (2) $c$ and $c'$ are identical, do
7.3.1 $TC = TC$ + *graphTransformCost*($G_c$, $G_{c'}$);
7.3.2 $L = L - \{c\}$; $L' = L' - \{c'\}$; //Since $(c, c')$ have been compared, they are deleted from their corresponding lists
7.4 For each pair $(c, c')$ satisfying: (1) $c \in L$, and $c' \in L'$; and (2) $c$ and $c'$ are not identical but are about the same topic, do
// Transformation cost is incremented based on the partial similarity of $c$ and $c'$
7.4.1 $TC = TC + W(c) * (1 - [Attributes(c) \cap Attributes(c')] / [Attributes(c) \cup Attributes(c')])$;
7.4.2 $TC = TC$ + *graphTransformCost*($G_c$, $G_{c'}$);
7.4.3 $L = L - \{c\}$; $L' = L' - \{c'\}$;
7.5 For each node $v \in L$, do // The nodes left in $G$ after steps 7.3 and 7.4 are deleted
7.5.1 $TC = TC$ + *subGraphDeleteCost*($G_v$);
8. Return $TC$;
}

// An utility function to calculate the edit cost of deleting a sub-graph $SG$
*real_number* ***subGraphDeleteCost***($SG$) {
1. Let $r$ represent the root of $SG$; Let $ac = W(r)$; // $ac$ is a local variable to accumulate the cost to delete the sub-graph $SG$
2. For each node $v \in SG$ and $v$ is a child node of $r$, do
2.1 $ac = ac + W(v)$ + *subGraphDeleteCost*($G_v$); // $G_v$ is the subgroup of $SG$ rooted at $v$
3. Return $ac$;
}

**Figure 7**: Algorithm to Calculate the Dissimilarity of Two Component Structures

*c* and *c*', are identical, no edit is necessary to substitute *c* with *c*'. Therefore, no cost is accumulated in graph transformation in terms of *c* and *c*'. However, if *c* and *c*' are partially similar (i.e., they are not identical but about the same topic), the edit cost of substituting *c* with *c*' depends on how similar *c* and *c*' are. More specifically, the cost of substituting a node *c* with another node *c*' is calculated as $W(c) * (1 - [Attributes(c) \cap Attributes(c')] / [Attributes(c) \cup Attributes(c')])$, where *Attributes*(*c*) and *Attributes*(*c*') represent the set of attributes that node *c* and *c*' represent, respectively. Finally, for every node *v* left in $G_1$, since *v* has no similarity (i.e., neither identical nor about the same topic) with any node of $G_2$, *v* should be deleted.

As shown in Algorithm 3, the final component structural dissimilarity between two compound objects $O_1$ and $O_2$, denoted as *structureDissimilarity*($O_1$, $O_2$), is calculated using Formula (7), where $TC_{1\_2}$ represent the edit cost to transform $G_1$ toward $G_2$ and $TC_{2\_1}$ represents the edit cost of the opposite transformation from $G_2$ toward $G_1$; and $W(c_i)$ and $W(c_j)$ represent the weights assigned to each node $c_i$ and $c_j$ of $O_1$ and $O_2$, respectively.

$$structureDissimilarity(O_1, O_2) = \frac{(TC_{1\_2} + TC_{2\_1})}{\sum_{c_i \in G_1} W(c_i) + \sum_{c_j \in G_2} W(c_j)} \quad (7)$$

The general formula to calculate the combined trust *t* from the trust values of *n* objects, i.e., $t_1$, $t_2$, …, $t_n$, is shown as $t = 1 - (1 - t_1) * (1 - t_2) * \ldots * (1 - t_n)$ (Zuo & Panda, 2006). As we can see, the combined trust value is higher than any of the individual object's trust value.

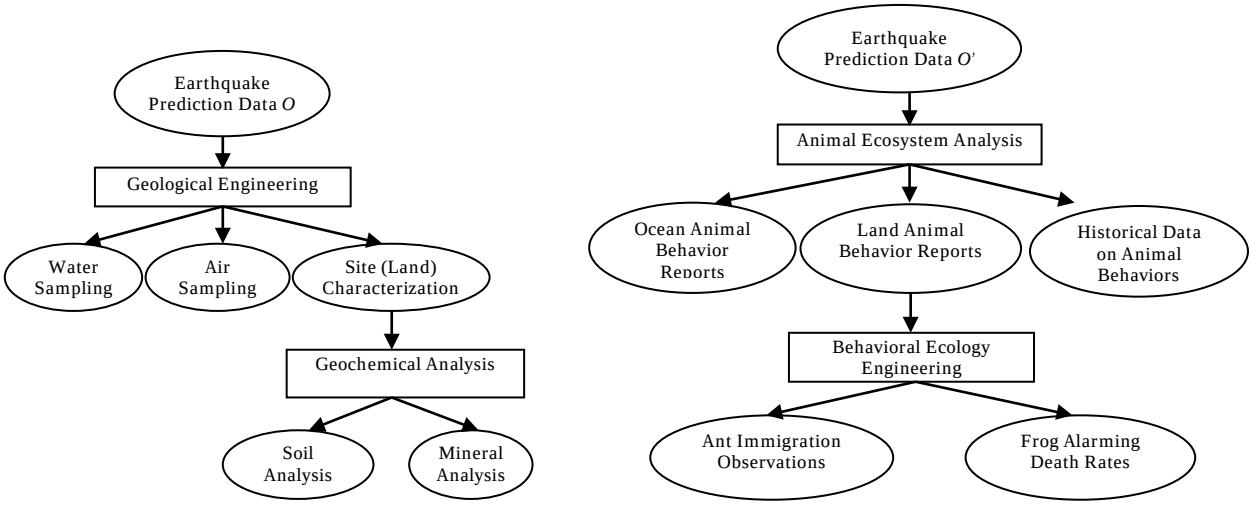


**Figure 8**. Component Graphs of the Objects Representing the Island Earthquake Prediction Data - Left: Result Obtained Using Geological Approaches and Right: Result Obtained Based on Animal Behavior Analysis.

With respect to our earthquake example discussed in Section 4.2.1, the component graphs of the two objects *O* and *O*' (about the earthquake prediction) are given in Figure 8. It is assumed that the earthquake prediction results announced by the two groups of scientists can be separately evaluated with trust values of $t_1 = 0.8$ and $t_2 = 0.6$, respectively. As we know that an accurate earthquake prediction is hard, if not impossible, and people would like to see more evidence to confirm their observations. By looking at the two sets of results together, an evaluator would improve his/her confidence on the prediction announcements. Using Algorithm 3, the similarity between the two component graphs as shown in Figure 8 is calculated as zero, i.e., *structureDissimilarity*(*O*, *O*') = 0. This is because the two groups of scientists

used completely different approaches to predict the earthquake. Hence, the two objects have totally dissimilar component structures. Since their predictions present very similar results about the possibility of an earthquake during the next month but they have used completely different means to reach their conclusions, the individual trust values can be elaborated to a higher level. According to the trust combination model, the combined trust value of the similar results provided by the two group of scientists is calculated as $t = 1 - (1 - 0.8) * (1 - 0.6) = 0.92$. This shows a great improvement in the user's confidence towards the results obtained from multiple sources on the same topic.

## 5. CONCLUSION, IMPLICATION, LIMITATION AND FUTURE WORK

*Conclusion*: In a virtual organization (VO), each participant needs to carefully evaluate and select shared objects for quality assurance. Wrong or incomplete information could result in a wrong decision or cause undesirable consequence. In this paper, we propose a trust-based framework to facilitate object evaluation. Current research on trust management focuses at the subject level and there is a lack of formal approaches to evaluate the trustworthiness of an object. Our work bridges this gap by proposing a framework for object trust management. We discussed two object trust principles as well as the formal models to apply those principles in a VO environment. The principles specify rationale and guidelines for object trust assessment. VO participants can then evaluate and select the objects with the required level of quality and security features. Studying the trustworthiness of an object requires the evaluator to possess solid domain knowledge about the object. Our proposed principles and models help users apply general rules and systematic approaches to guide them in object evaluation and selection.

*Implication of the research to knowledge management*: The power of information sharing strengthens organizations and has led to the science of knowledge management. One of the main challenges in knowledge management is to assess the quality and security features (in terms of software programs) of the large amount of knowledge elements. This paper provides theoretical and empirical insights about how object trust can be composed and combined. It has a strong implication in identifying, creating, evaluating, distributing, and using quality intelligent items. The proposed approach offers an effective tool for knowledge evaluation and thus contributes to the knowledge management in general and in a VO specifically.

*Limitation of the research and future work*: Since the paper uses a component-based approach to evaluate the quality and security of an object, the methodology we propose is only applicable to object-oriented or component-oriented software programs and other similar types of knowledge elements. Furthermore, this research is mainly based on formal principles, theoretical approaches and computational algorithms. In our future work, we plan to use empirical components such as more comprehensive case scenarios, model prototyping, and user experience analysis to further validate the proposed approaches and to quantify their benefits to information sharing and knowledge management in virtual organizations.

## ACKNOWLEDGMENT

Yanjun Zuo's work is partially supported by the US Air Force Office of Scientific Research (AFOSR) under Award FA9550-12-1-0131. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of AFOSR.

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions, which made this paper stronger.

## REFERENCES

Akram, K., Siddiqui, S. H., Nawaz, M. A., Ghauri, T. A. and Cheema, A. K. H. 2011. "Role of Knowledge Management to Bring Innovation: An Integrated Approach", *International Bulletin of Business Administration*, No. 11, pp. 121-134.

Alcalde, B., Dubois, E., Mauw, S., Mayer, N. and Radomirovic, S. 2009. "Towards a Decision Model Based on Trust and Security Risk Management", *Proc. of Australasian Information Security Conference*, p. 10, Wellington, New Zealand.

Biba, K. 1975. "Integrity Considerations for Secure Computing Systems," Mitre Report MTR-3153, Mitre Corporation, Bedford, MA, USA.

Blaze, M., Feigenbaum, J. and Lacy, J. 1996. "Decentralized Trust Management," *Proc. of 17th Symposium on Security and Privacy*, pp. 164-173, Oakland, USA.

Bouthillier, F. and Shearer, K. 2002. "Understanding Knowledge Management and Information Management: the Need for an Empirical Persective", *Information Research*, Vol. 8, No. 1.

Bray, D. and Konsynski, B. 2007. "Virtual worlds: Multi-disciplinary Research Opportunities," *ACM SIGMIS Database*, 38 (4), pp. 17–25.

BRR, 2005, Business Readness Rating Model avaialabe at http://www.openbrr.org (accessed on March 8, 2012 and is currently being updated).

Canfora, G, Cimitile, A. and Visaggio, C. A. 2002. "From Knowledge Management Concepts toward Software Engineering Practices," *Proc. of 4th International Conference on Product Focused Software Process Improvement*, pp. 407-422, Rovaniemi, Finland.

Chu, Y.-H., Feigenbaum, J., LaMacchia, B., Resnick, P. and Strauss, M. 1997. "REFEREE: Trust Management for Web Applications," *Computer networks and ISDN Systems*, Vol. 29, pp. 8-13.

Clemons, E. Gao, G. and Hitt, L. 2006. "When Online Reviews Meet Hyper Differentiation: a Study of the Craft Beer Industry," *Journal of Management Information Systems*, 23 (2), pp. 149–171.

Cody, E., Sharman, R., Rao, R. and Upadhyaya, S. 2008. "Security in grid computing: a review and synthesis," *Decision Support Systems*, 44 (4), pp. 749–764.

Dellarocas, C. 2003. "The Digitization of Word-of-Mouth: Promise and Challenges of Online Reputation Systems," *Management Science*, 49 (10), pp.1407–1424.

Dubois, D. and Prade, H. 1987 "The Principle of Minimum Specificity as a Basis for Evdiential Reasoning", Lectures Notes in Comptuer Science, Vol. 286/1987, pp. 75-84.

Earle, TC. 2010. "Trust in Risk Management: A Model-Based Review of Empirical Research", *Risk Analysis: The Official Publication of the Society for Risk Analysis*, 30(4), pp. 541-574.

Elliott, M. and Scacchi, W. 2004. "Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture," *Free/Open Source Software Development*, Idea Publishing.

Gligor, V. and Wing, J. 2011. "Towards a Theory of Trust in Networks of Humans and Computers", *Proc. of 19th Internaitonal Workshop on Security Protocols*, p. 22, Cambridge, UK.

Gross, R. and Acquisti, A. 2005. "Information Revelation and Privacy in Online Social Networks (The Facebook Case)," *Proc. of ACM Workshop on Privacy in the Electronic Society*, Alexandria, VI, USA.

He, W., Fang, Y. and Wei, K. 2009. "The Role of Trust in Promoting Organizational Knowledge Seeking Using Knowledge Management Systems: An Empirical Investigation", *Journal of the American Society for Information Science and Technology*, 60(3), pp. 526-537.

Jaccard Index, 2013, http://en.wikipedia.org/wiki/Jaccard_index (accessed on January 10, 2013).

Jason, A., Scott, B. and LePine, J. 2007. "Trust, Trustworthiness, and Trust Propensity: A Meta-Analytic Test of Their Unique Relationships with Risk Taking and Job Perfromance", *Journal of Applied Psychology*, 92(4), pp. 909-927.

Josang, A. and Tran, N. 2000 "Trust Management for E-Commerce," *Proc. of Virtual Banking*, a virtual conference available at http://virtualbanking2000.com (accessed March 8, 2012).

Kahn, B., Strong, D., Wang, R. 2002. "Information Quality Benchmarks: Product and Service Performance," *Communications of ACM*, 45(4), pp. 184-192.

Kang, Y., Williams, L, Clark, M., Gray, J. and Bargh, J. 2011. "Physical Temprature Effects on Trust Behavior: the Role of Insula", *Social Cognitive and Affective Neuroscience*, 6(4), p. 9.

Khare, R. and Rifkin, A. 1998. "Trust Management on the World Wide Web," *Computer networks and ISDN Systems*, Vol. 30, pp. 1-7.

Kim, D., Ferrin, D. and Rao, H. 2008. "A Trust-Based Consumer Decision-Making Model in Electronic Commerce: The Role of trust, perceived Risk, and their Antecedents", *Decision Support Systems*, Vol. 44, pp. 544-564.

Kim, Y. and Phalak. R. 2012. "A Trust Predication Framework in Rating-Based Experience Sharing Social Networks without a Web of Trust", *Information Sciences*, Vol. 191, pp. 128-145.

Lacohee, H., Phippen, A. and Furnell, S. 2006. "Risk and Restitution: Assessing How Users Establish Online Trust", *Computers and Security*, Vol. 25, pp. 486-493.

Lim, K. H., Sia, C. L., Lee, M. K. O., and Benbasat, I. 2006. "Do I trust You Online, and If So, Will I Buy? An Empirical Study of Two Trust-Building Strategies," *Journal of Management Information Systems*, 23(2), pp. 233–266.

Lui, H., Lim, E. P., Lauw, H. W., Le, M. T., Sun, A., Srivastava, J., and Kim, Y. A. 2008. "Predicting Trusts Among Users of Online Communities - an Epinions Case Study," *Proc. of the Ninth International ACM Conference on Electronic Commerce*, pp. 97-118, Chicago, IL, USA.

Ma, Z., Krings, A. and Hung, C. 2009. "The Handicap Principle for Trust in Computer Security, the Semantic Web and Social Networking", *Proc. of the International Conference on Web Information Systems and Mining*, pp. 458-468.

Mann, B. Some Data Derivation and Provenance Issues in Astronomy, Retrieved on December 1, 2012 at: http://people.cs.uchicago.edu/~yongzh/papers/mann.ps.

Marsh, S. P. 1994. "Formalizing Trust as a Computational Concept," Ph.D. Dissertation, University of Stirling.

Meyer, S., Ward, P., Coveney, J. and Rogers, W. 2008. "Trust in the Health Systems: an Analysis and Extension of the Social Theories of Giddens and Luhmann", *Health Socilogy Review*, Vol. 17, pp. 177-186.

Michael, J. B. and Gaines, L. T. 2001. "Trust Management in Distributed Databases," *Chapter 29, Data and Applications Security: Developments and Directions*, Kluwer.

Mollering, G. 2001. "The Nature of Trust: From Georg Simmel to a Theory of Expectation, Interpretation and Suspension", *Socilogy*, 35(2), pp. 403-420.

Pavlou, P.A., and Gefen, D. 2004. "Building Effective Online Marketplaces with Institution-based Trust," *Information Systems Research*, 15(1), pp. 37–59.

Perer, A. and Shneiderman, B. 2006. "Balancing systematic and flexible exploration of social networks, *IEEE Transactions on Visualization and Computer Graphics*", 12 (5), pp. 693–700.

Piao, C. and Gan X. 2009. "Research on Trust Management Model for E-Commerce Based on Fuzzy Clustering Method", *Proc. of IEEE International Conference on e-Business Engineering*, pp. 188-195.

Ristenpart, T., Tromer, T., Shacham, H. and Savage, S. 2009. "Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds," *Proc. of 16th ACM Conference on Computer and Communications Security*, pp. 199-212, Chicago, IL, USA.

Ronald, W. and Lawrence, D. 2007. "Organizational Trust, Trust in Chief Executive and Work Satisfication", *Public Peronnel Management*, 36(2), pp. 165-179.

Sarrafzadeh, M., Martin, B. and Hazeri, A. 2006. "LIS Professional and Knowledge Management: Some Recent Perspectives", Library Management, 27(9), pp. 621-635.

Squicciarini, A., Bertino, E., Ferrari, E., and Ray, I. 2006. "Achieve Privacy in Trust Negotiations with an Ontology-Based Approach," *IEEE Transactions on Dependable and Secure Computing*, 3(1), pp. 13-30.

Squicciarini, A. Bertino, E., Ferrari, E., Paci, F. and Thuraisingham, B. 2007. "PP-Trust- *X*: A System for Privacy Preserving Trust Negotiations", *ACM Transactions on Information and Systems Security*, 10(3), p. 50.

Scacchi, W. 2007. "Understanding the Development of Free E-Commerce/E-Business Software: A Resource-Based View," *Emerging Free/Open Source Software Practices*, IDEA group Publishing, Hershey, PA, USA.

Sherif, K., Munasinghe, M. and Sharma, C. (2012). "The Combinative Effect of Electronic Open Networks and Closed Interpsonal Networks on Knowledge Creation in Academic Communities", *VINE – The Journal of Information and Knowledge Management Systems*, 42(2), pp. 277-294.

Soh, C., Markus, M.L. and Gho, K. 2006. "Electronic Marketplaces and Price Transparency: Strategy, Information Technology, and Success", *MIS Quarterly*, 30(3), pp. 705-723.

Son, J. and Benbasat, I. 2007. "Organizational Buyers' Adoption and Use of B2B Electronic Marketplaces: Efficiency and Legitimacy Oriented Perspectives," *Journal of Management Information Systems*, 24(1), pp. 55-99.

Turel, O., Yuan, Y., and Connelly, C. E. 2008. "In Justice We Trust: Predicting User Acceptance of E-Customer Services," *Journal of Management Information Systems*, 24 (4), pp. 123–151.

Wang, R., Kon, H., and Madnick, S. 1993. "Data Quality Requirements Analysis and Modeling," *Proc. of 9th International Conference of Data Engineering*, Vienna, Austria.

Ward, P. and Meyer, S. 2009. "Trust, Socail Quality and Wellbeing: A Sociological Exegesis", *Development and Society*, 38(2), pp. 339-363.

Wickramasinghe, V. and Widyaratne, R. 2012. "Effects of Interpresonal Trust, Team Leader Support, Rewards, and Knowledge Sharing Mechanisms on Knowledge Sharing in Project Terms", *VINE – The Journal of Information and Knowledge Management Systems*, 42(2), pp. 214-236.

Zhang, Y., Chen, H., Xia, C. and Jiang, X. 2010. "Building Trust in Electronic Communities by Mining Web Content", *International Journal of Computational Science and Engineering*, 5(1), pp. 58-67.

Zuo, Y. and Panda, B. 2006. “Information Trustworthiness Evaluation Based on Trust Combination,” *Proc. of the 21th Annual ACM Symposium on Applied Computing*, pp. 1880-1885, Dijon, France.

Zuo, Y. 2007. “Component Comparison Based Information Quality Assurance”, *International Jornal of Information Quality*, 1(3), pp. 295-313.

Zuo, Y. and Panda, B. 2008. “Two-Level Trust-Based Decision Model for Information Assurance in a Virtual Organization”, *Decision Support Systems*, 45(2), pp. 291-309.