



Trustworthiness requirement-oriented software process modeling

Xuan Zhang^{1,2†} | Xu Wang^{3†} | YanNi Kang¹

¹School of Software, Yunnan University, Kunming, Yunnan, China

²Key Laboratory of Software Engineering of Yunnan, Kunming, Yunnan, China

³School of Economics, Yunnan University, Kunming, Yunnan, China

Correspondence

Xuan Zhang, School of Software, Yunnan University, Kunming, Yunnan 650091, China. Email: zhxuan@ynu.edu.cn

Funding information

National Natural Science Foundation of China, Grant/Award Numbers: 61862063, 61502413, 61262025, 61379032 and 61662085; National Social Science Foundation of China, Grant/Award Number: 18BJL104; Natural Science Foundation of Yunnan Province, Grant/Award Number: 2016FB106; Natural Science Foundation of Yunnan Educational Committee, Grant/Award Number: 2015ZQ20; Natural Science Foundation of Key Laboratory of Software Engineering of Yunnan Province, Grant/Award Number: 2015SE202; Data-Driven Software Engineering Innovative Research Team Funding of Yunnan Province, Grant/Award Number: 2017HC012; Software Engineering Innovative Research Team Funding of Yunnan University

Abstract

Trustworthy software is delivered by enacting trustworthy software processes. The purpose of this paper is to propose an approach to modeling trustworthiness requirement-oriented software processes. First, based on the aspect-oriented modeling techniques, separation of concerns is used to separate the crosscutting activities and the core activities according to the different trustworthiness requirements and functional requirements. A goal-oriented modeling and reasoning method for trustworthiness requirements to find the crosscutting activities that satisfy multiple trustworthiness requirements is presented. Then, base processes are modeled for functional requirements. The crosscutting activities for trustworthiness requirements are decomposed into processes or tasks and encapsulated in aspects that are woven into the base processes. In the weaving procedure, correct weaving methods between multiple aspects and between aspects and base processes are designed. Errors or mistakes of aspect-oriented process modeling are prevented. Finally, trustworthy third-party certification authority software is studied systematically in a case study, and performance evaluations are conducted to show the cost and effect of the approach.

KEYWORDS

aspect-oriented modeling, goal-oriented modeling, Petri nets, software process modeling, software trustworthiness requirement

1 | INTRODUCTION

Our daily lives and industrial processes are heavily reliant on a wide range of underpinning software. Therefore, addressing the trustworthiness of software is a pressing need. Software development and evolution are process-intensive undertakings. Delivering trustworthy software requires the enactment of trustworthy software processes.¹ The degree of confidence that a software process produces expected trustworthy products that satisfy their requirements is also critical and is called the process trustworthiness.¹ In this paper, we focus on how to model a trustworthy software process and ensure the process trustworthiness.

The issues that motivated our study include the following:

- Current techniques focus on the implementation of individual requirements for trustworthy software. In addition, many of the techniques required to enforce one non-functional requirement (NFR) of trustworthy software undermine the enforcement of others.² Techniques that support the development of systems that satisfy multiple requirements are needed.

[†]These authors contributed equally to this work.

- Traditional software process activities are designed for functional requirements. For trustworthy software requirements, the specified activities must be provided and integrated into the traditional software processes. Additionally, these two types of activities should not be tangled in the software processes. Because of the volatile requirements, flexible and maintainable process modeling is also required.
- When integrating trustworthy software activities into a software process, the execution of the process must not be interrupted, and the trustworthy software activities must validly provide the expected trustworthiness capability. Therefore, ensuring the correctness of the integrated process models is a prerequisite for process trustworthiness.

An approach to modeling the trustworthiness requirement-oriented software process (TROSP) and the corresponding methods for ensuring the correctness of the modeling are presented in this paper. Petri net extended with aspect-oriented modeling is used as the process modeling language. The analysis techniques of Petri nets are applied to analyze the correctness of the modeling. Compared with other process-modeling languages, Petri nets provide the following advantages:

- Petri nets are well-founded process modeling techniques. A process specified in Petri net languages has a clear and precise definition since the semantics of the Petri nets have been formally well defined.³
- Causal dependencies and interdependencies in some sets of events are represented explicitly.
- Petri net representations allow model properties to be verified and correctness proofs to be performed in a specific way.⁴
- Petri nets serve more as a modeling language and an analysis or verification tool, and its main contribution is to cybernetics. The motivation of research on software cybernetics is to explore whether and how software behavior can be controlled in the software, including software processes and software systems.⁵

Separation of concerns is used to separate crosscutting and core activities. In this paper, crosscutting activities are trustworthy software activities, whereas core activities are traditional activities for functional requirements. An aspect-oriented paradigm is used to provide a proper mechanism for modularization, thus reducing the model complexity while improving flexibility and maintainability.

The remainder of this paper is organized as follows. Section 2 introduces the method for obtaining trustworthy software activities through goal-oriented modeling and reasoning. Section 3 details the framework and workflow for modeling TROSPs. In Section 4, we illustrate a case study for modeling the processes of trustworthy third-party certification authority software. Section 5 evaluates the method by comparing effectiveness and efficiency and describes the limitations of the method. Section 6 describes related work. Section 7 concludes with the contributions of our work.

2 | GOAL-ORIENTED MODELING AND REASONING FOR TRUSTWORTHINESS REQUIREMENTS

Trustworthy software is required in a variety of systems—military, aviation, transportation, financial, medical, etc.—and must satisfy a variety of requirements.² In Trusted Software Methodology,⁶ software trustworthiness is the “degree of confidence that the software satisfies its requirements.” Therefore, trustworthy software is defined as software that satisfies the range of trustworthiness objectives established based on its requirements.¹

Requirements for trustworthy software consist of both functional requirements and NFRs. Since goal-oriented models have been widely advocated for early requirements modeling, we use the term hard goal to denote a functional requirement and the terms trustworthiness goal and soft goal to denote NFRs. Hard goals are functional requirements that must be strictly implemented as the most critical requirements for trustworthy software. Non-functional requirements of trustworthy software are divided into trustworthiness goals and soft goals. Trustworthiness requirements (TRs) are a specific set of trustworthiness attributes for a particular trustworthy software development project. In other words, TRs are circumstance-dependent requirements that satisfy the stakeholders' set of trustworthiness expectations. Depending on the stakeholder's value propositions and negotiation results, TRs are part of the NFRs, such as reliability, safety, security, portability, and maintainability. Soft goals are not trustworthiness goals but are related to the qualities of the trustworthy software. The hard goals and trustworthiness goals are combined to form the TRs, as depicted in Figure 1. Since poor quality software is not trustworthy software, the soft goals define the trustworthy software quality requirements.

The understanding of trustworthiness goals has differed over time. Table 1 is an extension of the table from Zhang's attributes of trustworthy software systems.²⁰

As shown in Table 1, trustworthiness is a holistic property, which means that the methods for building trustworthy software must satisfy multiple trustworthiness goals.

In a specific software project, the stakeholders are the providers and evaluators of the requirements. However, the requirements of the stakeholders vary according to their different roles, responsibilities, and experiences. The interactive and iterative Delphi method²¹ is useful for reconciling the trustworthiness goals of multiple stakeholders and achieving consensus.

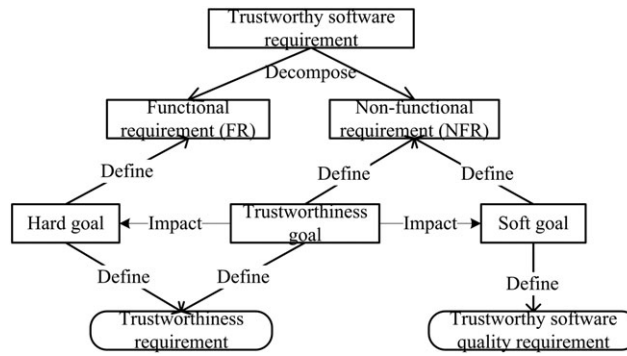


FIGURE 1 Goal-oriented trustworthy software requirement

TABLE 1 The understanding of trustworthiness goals

Source	Trustworthiness Goals
TCSEC (trusted computer system evaluation criteria) ⁷	Security (confidentiality, authenticity, accountability)
Dependability in IEC 60050-192 ⁸	Reliability, availability, recoverability, maintainability, maintenance support performance, durability, safety, security
TSM (Trusted Software Methodology) ⁶	Security, reliability (availability)
Microsoft ^{9,10}	Reliability, security, privacy, business integrity
TCG (Trustworthy Computing Group) ¹¹	Security, maintainability
Littlewood, Schmidt ^{12,13}	Reliability, safety, robustness, availability, security
DARPA's CHATS (Defense Advanced Research Projects Agency's composable high-assurance trustworthy systems) ¹⁴	Security (integrity, confidentiality, authentication, authorization, accountability), reliability (fault tolerance), performance (time-behavior, capacity), survivability
NSS2 (the second National Software Summit) ¹⁵	Security, safety, reliability, survivability, performance
TrustSoft (TrustSoft Graduate School in University of Oldenburg) ^{16,17}	Correctness, safety, quality of service (performance, reliability, availability), security, privacy
COMPSAC (International Computer Software and Applications Conference) ¹⁸	Availability, reliability, security, survivability, recoverability, confidentiality, integrity
ICSP (International Conference on Software Process) ¹	Functionality, reliability, safety, usability, security, portability, maintainability
Trustie-STC 1.0 (Software Trustworthiness Classification Specification) ¹⁹	Availability, reliability, security, real time, maintainability, survivability

Once the software trustworthiness goals are agreed upon, the problem of how to ensure trustworthiness must be addressed. Two possible approaches exist. In the first approach, the emphasis is on techniques for developing and examining software products directly. In the second approach, the emphasis is on the processes used to create these products.⁶ The primary benefit of focusing on techniques is that abundant useful information that is familiar to most programmers, engineers, and managers can be easily obtained. However, the emphasis on process complements these techniques by focusing on the manner in which they are performed during the actual development lifecycle. In addition, a process emphasis allows for the reuse of valuable previous experiences.

In the following, from the process perspective, goal-oriented modeling is used to find trustworthiness goal-oriented activities (TG_activities) to model TROSPs.

2.1 | Trustworthiness goal-oriented modeling

Trustworthy software is required in a variety of systems filtering numerous studies, including Lyu's and Musa's software reliability engineering,^{22,23} Anderson's and Howard's security engineering,^{9,10,24} Ericson's and US DoD's safety engineering,^{25,26} Nielsen's usability engineering,²⁷ Boehm's process strategies,²⁸ and security-related activities in the software assurance maturity model.²⁹ Table 2 lists these TG_activities.

The enforcement of each TG_activity implements a specific trustworthiness goal. However, this enforcement may undermine the other trustworthiness goals or soft goals. For example, the enforcement of a *Fault-Tolerance Design* activity for reliability improves usability but undermines functional correctness. Therefore, we need to find a set of TG_activities that satisfies multiple goals, including trustworthiness goals and soft goals. By borrowing concepts from goal-oriented requirements modeling, a trustworthiness requirements meta-model (TRMM) is designed, as shown in Figure 2.

TABLE 2 Trustworthiness goals-oriented activities

Trustworthiness Goals	TG_Activities
Functional completeness and correctness	Analyze and assess user characteristics, analyze and assess software context, functionality analysis, validation and verification of requirements, validation and verification of design, data type checking, proof of program correctness, code review.
Reliability	Define functional profile, define and classify failure, reliability requirement identification and acquisition, reliability modeling, reliability analysis, fault tree analysis, reliability forecasting, reliability testing of off-the-shelf and outsourcing software, reliability planning, reliability deploying, use software reliability design criteria, fault/failure forecasting design, fault prevention design, error correction design, fault tolerance design, redundancy design, maintainability design, reliability-optimized design, determine the engineering measures for reliability goals, software reliability growth modeling, define resource deployment based on functional profile, failure testing, software reliability growth testing, stress testing, regression testing, software reliability verification, create maintenance scheme, software reliability monitoring, satisfaction tracking of reliability, continuous process improvement, reliability measurement, reliability engineering management, management of failure introduction and propagation, reliability management of off-the-shelf and outsourcing software, establish incident response plan, execute incident response plan.
Safety	Software safety plan, establish SSP (system safety program), identify hazards, assess hazard risk, document SSP, determine acceptable hazard level, use hazard mitigation methods, determine order of hazard mitigation precedence, proof of program correctness, redundancy design, validation and verification of risk reduction, hazard and risk review, establish HTS (hazard tracking system), establish HARs (hazard action records), safety monitoring.
Security	Define minimum security criteria, create quality gates/bug bars, perform security and privacy risk assessment, define misuse cases, security asset and boundary identification, security strategy identification, security techniques assessment, integrate security analysis into resource management process, establish security and privacy design requirements, perform security and privacy design review, code review, satisfy minimum cryptographic design requirements, analyze attack surface, threat modeling, create incident response plan, security testing, deprecate unsafe functions, static analysis, dynamic analysis, fuzz testing, penetration testing, attack surface review, execute incident response plan, vulnerability management.
Accuracy	Accuracy boundary identification, define and classify accuracy, solve inaccuracy, reconstruction from approximation to accuracy, accuracy design, document accuracy requirements.
Maintainability	Maintenance design, cohesion design, abstraction design, more access interface design, maintainability assessment, create maintenance scheme, collect and analyze software execution data, continuous process improvement, maintenance qualities management.
Compatibility	Collaboration analysis, financial impact analysis, compatibility modeling and forecasting, interaction interface design, multi-domain architecture design, compatibility test.
Usability	User characteristics analysis, functionality analysis, competitive analysis, financial impact analysis, parallel design, participatory design, coordinated design of the interface, iterative design, more options design, fault-tolerance design, heuristic analysis, prototyping, empirical testing, collect feedback from field use.
Performance efficiency	Determine performance baseline, performance modeling and analysis, prototyping, performance simulation, performance tuning, user coordinated testing.

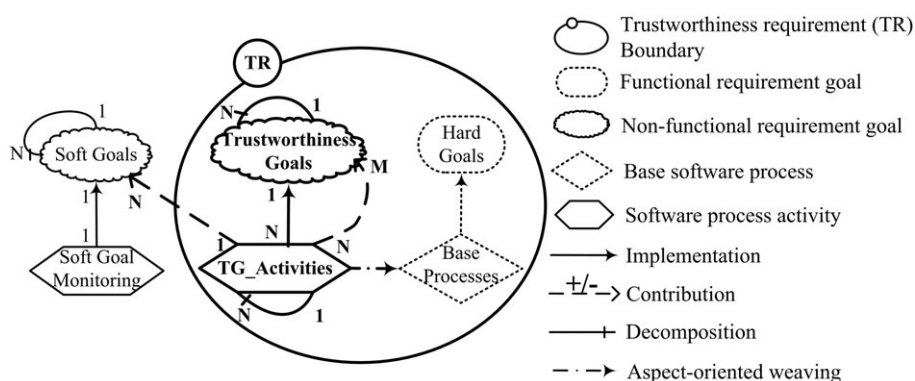


FIGURE 2 Trustworthiness requirements meta-model (TRMM)

In the TRMM, the TR boundary shows that the emphasis of the model is the TRs. The TG_activities are enforced to implement trustworthiness goals. As mentioned above, the enforcement of some TG_activities may undermine some other trustworthiness goals. In Figure 2, the contribution links represent these relations and connect the TG_activities and the affected trustworthiness goals with dashed lines. Since the soft goals are trustworthy software quality requirements, the effects on the soft goals are also depicted with dashed lines. In addition, the *Soft Goal Monitoring* activity implements the quality monitoring for each soft goal to avoid the poor quality that may be caused by the TG_activities. Based on the

decompositions of the NFRs, the trustworthiness and soft goals can be decomposed into subgoals. The base processes are the processes that implement the hard goals and will be integrated with the TG_activities. Formally, we define the TRMM as follows:

Definition 1. (TRMM). A TRMM is a 2-tuple $TRMM = (N, R)$, where:

(1) $N = TG \cup SG \cup TA$ is a set of nodes, where

- TG is a set of trustworthiness goals, and $TG \neq \emptyset$;
- SG is a set of soft goals;
- TA is a set of trustworthiness goal-oriented activities (TG_activities), and $TA \neq \emptyset$;

(2) $R = R^{dec} \cup R^{imp} \cup R^{ctr}$ is a set of binary relations of nodes, where:

• $R^{dec} \subseteq (TG \times TG) \cup (SG \times SG) \cup (TA \times TA)$ is a decomposition relations set, where:

- $TG \times TG = \{(tg, tg') \mid tg, tg' \in TG \wedge tg \text{ decompose into } tg', \text{ and } tg' \text{ is a subnode of } tg\}$;
- $SG \times SG = \{(sg, sg') \mid sg, sg' \in SG \wedge sg \text{ decompose into } sg', \text{ and } sg' \text{ is a subnode of } sg\}$;
- $TA \times TA = \{(ta, ta') \mid ta, ta' \in TA \wedge ta \text{ decompose into } ta', \text{ and } ta' \text{ is a subnode of } ta\}$;

• $R^{imp} \subseteq TG \times TA$ is an implementation relation set between the trustworthiness goal set TG and the TG_activity set TA ; $R^{imp} = \{(tg, ta) \mid tg \in TG \wedge ta \in TA \wedge ta \text{ implements } tg\}$;

• $R^{ctr} \subseteq (TA \times TG) \cup (TA \times SG)$ is a TG_activities' contribution relation set, that is, in relation to a trustworthiness goal set TG or a soft goals set SG ; $\forall r \in R^{ctr}, r \mapsto \{+, -\}$ (\mapsto is used as a mapping from a relation to a member of a set); $+$ is a positive contribution relation, whereas $-$ is a negative contribution relation:

- $TA \times TG = \{(ta, tg) \mid ta \in TA \wedge tg \in TG \wedge (ta, tg) \mapsto \{+, -\}\}$ are positive or negative contribution relations from a TG_activity set to a trustworthiness goal set;
- $TA \times SG = \{(ta, sg) \mid ta \in TA \wedge sg \in SG \wedge (ta, sg) \mapsto \{+, -\}\}$ are positive or negative contribution relations from a TG_activities set to a soft goal set;

(3) trm is a trustworthiness requirements model (TRM) that is modeled by using the TRMM.

When using the TRMM to model a TRM, constraints on the nodes and relations must be obeyed.

Constraint 1. (Node Constraint). Let $trm = (N, R)$ be a TRM, then $\text{dom}(r) = \{x \in N \mid \exists y \in N : (x, y) \in r, r \in R\}$, $\text{cod}(r) = \{x \in N \mid \exists y \in N : (y, x) \in r, r \in R\}$, and $\text{dom}(r) \cup \text{cod}(r) = N$.

Constraint 1 prevents all nodes in a TRM from being isolated. An isolated trustworthiness goal means that the modeling procedure is not complete because it must be implemented by at least one TG_activity. An isolated TG_activity means that the TRM is not correct because it must have at least one implementation relation to a trustworthiness goal. An isolated soft goal should be deleted because there is no contribution relation from any TG_activities, and thus it can be assumed to be satisfied.

Constraint 2. (Relation constraint). Let $trm = (N, R)$ be a TRM; $tg \in TG$ is a trustworthiness goal, $ta \in TA$ is a TG_activity, and $(ta, tg) \in R^{imp}$; there is no $(ta, tg) \in R^{ctr}$.

Constraint 2 states that if there is an implementation relation between a TG_activity and a trustworthiness goal, there is no contribution relation between them.

A base software process in the TRMM is a software process that is created for the functional requirements. It is an extended Petri net,³⁰ as defined in Definition 2 and 3. For the sake of simplicity, in the following, base process is used to denote base software process.

Definition 2. (Base process). A base process is a 4-tuple $p = (C, A; F, M_0)$, where (1) $(C, A; F)$ is a Petri net without isolated elements, $A \cup C \neq \emptyset$; (2) C is a finite set of conditions; (3) A is a finite set of activities; the execution of an activity is called that an activity fires or is enabled; (4) $F \subseteq (C \times A) \cup (A \times C)$ is the flow relation; $\text{dom}(F) \cup \text{cod}(F) = C \cup A$, where $\text{dom}(F) = \{x \in C \cup A \mid \exists y \in C \cup A : (x, y) \in F\}$, $\text{cod}(F) = \{x \in C \cup A \mid \exists y \in C \cup A : (y, x) \in F\}$; (5) M_0 is the initial marking, where a marking is a mapping $M: C \mapsto \{0, 1\}$.

A Petri net is a set of nodes and arcs. There are two types of nodes: places and transitions, which represent the state of the system and the occurrence of events, respectively. Arcs are directed and connect places with transitions or transitions with places. In Definition 2, a condition in a base process is a place in Petri nets. An activity is a transition, and a flow relation is an arc in Petri nets. The definitions in terms of conditions, activities, and flow relations are for a better understanding of software processes. The state of a base process is defined by a marking, which puts zero or one token (graphically represented by a dot) on each condition. The firing process induces a token's flow among conditions; when an activity fires, tokens from all its input conditions are moved to the activity output conditions. An activity can only be fired if there are tokens at its input conditions.

The activities in base processes, the TG_activities, and the soft goal monitoring activity are all the activities in software processes. The definitions of these activities are identical. Since we focus on the TG_activities, the definitions of these three types of activities coalesce formally into Definition 3.

Definition 3. (TG_Activity). A TG_activity is a 4-tuple $ta = (I, O, L, B)$, where (1) $I, O,$ and L are a set of input artifacts, a set of output artifacts, and a set of local artifacts, respectively, and (2) B , called the TG_activity body, is either a trustworthiness goal-oriented process (TG_process) or a set of trustworthiness goal-oriented tasks (TG_tasks) that operates on the artifacts $I, O,$ and L .

To integrate the TG_activities into base processes, aspect-oriented modeling method is used. When a TG_activity is decomposed into a TG_process or a set of TG_tasks, the TG_process or each TG_task is defined as an advice and encapsulated in an aspect. All of the encapsulated aspects are then woven into the base processes and base tasks. The details of the aspect-oriented modeling method will be described in Section 3.

Since the enforcement of a TG_activity may undermine the satisfaction of the other trustworthiness goals or soft goals, before defining and weaving the aspects into the base processes or base tasks, trustworthiness goal-oriented reasoning is performed.

2.2 | Trustworthiness goal-oriented reasoning

Let $trm = (N, R)$ be a TRM and $n \in N$ be a node in m ; four distinct status labels of nodes are introduced.

Definition 4. (Status Label). A status label is a first order predicate $SL(n) ::= SA(n) \mid PS(n) \mid PD(n) \mid DE(n)$, where $\forall n \in N$ (N is the set of nodes defined in Definition 1). $SA(n)$ denotes the satisfaction of node n . $PS(n)$ denotes the partial satisfaction of node n . $PD(n)$ denotes the partial denial of node n . $DE(n)$ denotes the denial of node n .

Given a TRM, propagation reasoning for contribution links occurs in two directions. One is the forward direction, which reasons from TG_activities to trustworthiness and soft goals, whereas the other is the opposing backward direction. These are called forward reasoning and backward reasoning. Forward reasoning starts with an analysis question of the form "How effective is a TG_activity with respect to the trustworthiness and soft goals in the model?".³¹ However, we want to be able to answer questions such as "If certain trustworthiness and soft goals need to be satisfied, what TG_activities in the model would satisfy these goals?" One way of using forward reasoning is to apply the procedure repeatedly and exhaustively for all reasonable TG_activities until either the desired values for goals are produced, or not. This method is tedious and laborious, especially for large models with many nodes.³¹ Conversely, backward reasoning starts with the trustworthiness and soft goals and works down the relations in the model to find potential satisfied TG_activities.

An early version of backward reasoning was described by Sebastiani et al.³² We have borrowed our general conjunctive normal form (CNF) formulation and part of the analysis predicates from this work. However, we made several modifications because the satisfied status of TG_activity n is only satisfaction $SA(n)$ or denial $DE(n)$.

Constraint 3. (TG_Activity Status Label Constraint). Let $trm = (N, R)$ be a TRM, $\forall ta \in TA$, and $TA \subseteq N$, then $SL(ta) ::= SA(ta) \mid DE(ta)$.

To formalize the propagation of satisfaction and denial evidence through the TRM, two fundamental axioms are introduced. The relation axioms and backward propagation axioms are presented in Table 3.

$$\text{Fundamental axiom : } \forall n \in N \quad SA(n) \rightarrow PS(n) \quad FA_1 \\ DE(n) \rightarrow PD(n) \quad FA_2$$

The fundamental axiom states that satisfaction and denial imply partial satisfaction and partial denial, respectively.

The backward reasoning encodes a TRM in CNF that iteratively applies a propositional satisfiability (SAT) solver to search for an acceptable solution. SAT is the problem of determining whether a Boolean formula ϕ admits at least one satisfying truth assignment to its input values.³² To express the problem of assigning status labels to a TRM in terms of a CNF SAT formula, we follow the formalization in Sebastiani et al.³² The SAT formula is constructed as follows:

$$\phi ::= \phi_{Initial} \wedge \phi_{Model} \wedge \phi_{Constraint} \wedge \phi_{Conflict} \quad (1)$$

$\phi_{Initial}$ is a representation of the input values that can be assigned to the trustworthiness goals and the soft goals. It is written in the form

$$\phi_{Initial} ::= \bigwedge_{n \in N} SL(n). \quad (2)$$

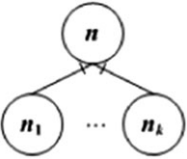
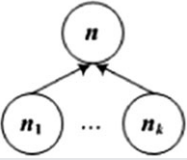
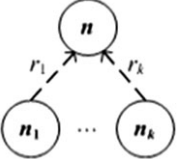
ϕ_{Model} is the representation of a TRM, given in the form

$$\phi_{Model} ::= \bigwedge_{n, n_i \in N} (\neg SL(n) \vee (\bigvee_{(n, n_i) \in R} SL(n_i))). \quad (3)$$

$\phi_{Constraint}$ and $\phi_{Conflict}$ are optional components. $\phi_{Constraint}$ allows the imposition of some constraints on the possible values of the nodes.

$$\phi_{Constraint} ::= \bigwedge_{n \in N} (LL(n) \mid \bigvee_{n \in N} LL(n)) \quad (4) \\ LL(n) ::= SL(n) \mid \neg SL(n), \neg SL(n) \rightarrow \{ \neg SA(n), \neg PS(n), \neg PD(n), \neg DE(n) \}$$

TABLE 3 Axioms for backward reasoning

Relation	Status	Relation Axioms	Backward Propagation Axioms
R^{dec} 	SA	$\bigwedge_{i=1}^k SA(n_i) \rightarrow SA(n)$	RA_1 $SA(n) \rightarrow \bigwedge_{i=1}^k SA(n_i)$
	PS	$\bigwedge_{i=1}^k PS(n_i) \rightarrow PS(n)$	RA_2 $PS(n) \rightarrow \bigwedge_{i=1}^k PS(n_i) (n_i \in TG \cup SG)$
	PD	$\bigvee_{i=1}^k PD(n_i) \rightarrow PD(n)$	RA_3 $PD(n) \rightarrow \bigvee_{i=1}^k PD(n_i) (n_i \in TG \cup SG)$
	DE	$\bigvee_{i=1}^k DE(n_i) \rightarrow DE(n)$	RA_4 $DE(n) \rightarrow \bigvee_{i=1}^k DE(n_i)$
R^{imp} 	SA	$\bigwedge_{i=1}^k SA(n_i) \rightarrow SA(n)$	RA_5 $SA(n) \rightarrow \bigwedge_{i=1}^k SA(n_i)$
	PS	$\bigvee_{i=1}^k SA(n_i) \rightarrow PS(n)$	RA_6 $PS(n) \rightarrow \bigvee_{i=1}^k SA(n_i)$
	PD	$\bigvee_{i=1}^k DE(n_i) \rightarrow PD(n)$	RA_7 $PD(n) \rightarrow \bigvee_{i=1}^k DE(n_i)$
	DE	$\bigwedge_{i=1}^k DE(n_i) \rightarrow DE(n)$	RA_8 $DE(n) \rightarrow \bigwedge_{i=1}^k DE(n_i)$
R^{ctr} 	SA	$r \mapsto \{+\} SA(n_i) \rightarrow SA(n)$	RA_9 $r_i \mapsto \{+\}$
	PS	$r \mapsto \{-\} SA(n_i) \rightarrow DE(n)$	RA_{10} $r_i \mapsto \{-\}$
	PD	/	$r_i \mapsto \{-\}$
	DE	$r \mapsto \{+\} (DE(n_i) \rightarrow DE(n)) \vee (DE(n_i) \rightarrow PD(n)) \vee (DE(n_i) \rightarrow SA(n)) \vee (DE(n_i) \rightarrow PS(n))$	RA_{11} $r_i \mapsto \{+\}$
			RA_{12} $r_i \mapsto \{-\}$
			BA_9 $SA(n) \rightarrow SA(n_i)$
			BA_{10} $SA(n) \rightarrow DE(n_i)$
			BA_{11} $PS(n) \rightarrow DE(n_i)$
			BA_{12} $PD(n) \rightarrow DE(n_i)$
			BA_{13} $DE(n) \rightarrow DE(n_i)$
			BA_{14} $DE(n) \rightarrow SA(n_i)$

When a SAT reasoning fails to find a satisfying assignment, it is useful to know about the underlying conflict (s).³¹ $\phi_{Conflict}$ is a representation of conflicts that we want to avoid. Formula (5) states that conflicts are not allowed; (6) states that n cannot be satisfied and partially denied and vice versa; (7) states that n can only be partially satisfied and partially denied. These three conflicts are called *strong conflict*, *medium conflict*, and *weak conflict*, respectively.

$$\phi_{Conflict} ::= \bigwedge_{n \in N} (\neg(PS(n) \wedge PD(n))) \quad (5)$$

$$\phi_{Conflict} ::= \bigwedge_{n \in N} (\neg(SA(n) \wedge PD(n)) \wedge \neg(PS(n) \wedge DE(n))) \quad (6)$$

$$\phi_{Conflict} ::= \bigwedge_{n \in N} (\neg(SA(n) \wedge DE(n))) \quad (7)$$

The most popular SAT algorithm is Davis Putnam Logemann Loveland, and zChaff³³ is probably the most efficient Davis Putnam Logemann Loveland implementation available.³² Using zChaff, a pseudo code implementing the backward reasoning is described in Algorithm 1.

Algorithm 1 SAT.

INPUT: $trm = (N, R)$.

OUTPUT: $Initial_Status(n_i)$, $SAT_Status(n_i)$, $Conflict(n_i)$.

BEGIN

$k_{TG} = |TG|$; $k_{TA} = |TA|$; $k_{SG} = |SG|$;

$\phi_{Model} := \top$; $\phi_{Initial} := \top$; $\phi_{Constraint} := \top$; $\phi_{Conflict} := \top$; /*The proposition \top represents the trivially true statement*/

FOR $i := 1$ TO $k_{TG} + k_{TA} + k_{SG}$ DO

BEGIN

Ask user to set $Initial_Status(n_i)$ for each node n_i ;

$\phi_{Initial} := \phi_{Initial} \wedge Initial_Status(n_i)$

END

$\phi_{Model} := \text{MakeSAT}(trm)$;

/*Create ϕ_{Model} from a TRM trm */

Ask user to input $\phi_{Constraint}$ and $\phi_{Conflict}$;

$\phi := \phi_{Model} \wedge \phi_{Initial} \wedge \phi_{Constraint} \wedge \phi_{Conflict}$;

Call prop2cnf.py to convert ϕ to CNF;

Call zChaff to calculate the satisfaction of CNF;

WHILE $RESULT = UNSAT$ DO

BEGIN

```

Output conflict clause;
Record  $SAT\_Status(n_i)$ ;
Ask user to decide whether the conflict is acceptable;
IF the conflict is allowed THEN
Modify  $\phi_{Conflict}$  in  $\phi$ 
ELSE
Modify  $\phi_{Initial}$  or delete clause of conflict nodes  $n_i$  in  $\phi$ 
 $\phi := \phi_{Model} \wedge \phi_{Initial} \wedge \phi_{Constraint} \wedge \phi_{Conflict}$ ;
Call prop2cnf.py to convert  $\phi$  to CNF;
Call zChaff to calculate the satisfaction of CNF
END;
List  $Initial\_Status(n_i)$ ,  $SAT\_Status(n_i)$ ,  $Conflict(n_i)$  for all nodes
END

```

MakeSAT (*trm*) in Algorithm 1 converts a TRM *trm* to ϕ_{Model} . After the user inputs $\phi_{Initial}$, $\phi_{Constraint}$, and $\phi_{Conflict}$, ϕ is generated by $\phi_{Model} \wedge \phi_{Initial} \wedge \phi_{Constraint} \wedge \phi_{Conflict}$. Then, a python code prop2cnf.py³⁴ converts ϕ to CNF form. By following the backward propagation axioms in Table 3, a modified zChaff is called to find a satisfying solution for CNF. We modified the zChaff to output the conflict clauses that the original zChaff does not output. If Algorithm 1 terminates and displays “UNSAT,” a conflict clause is output, and the corresponding $SAT_Status(n_i)$ of conflict node n_i is saved. If the conflict is acceptable, the user is asked to modify $\phi_{Conflict}$. Otherwise, the user is asked to modify $\phi_{Initial}$ or delete the conflicting node. The algorithm continues to execute until the result is “SAT,” which indicates that a satisfying solution has been found. In the following section, the TG_activities in this satisfied solution are integrated into the base processes.

3 | TRUSTWORTHINESS REQUIREMENT-ORIENTED SOFTWARE PROCESS MODELING

Li designed a software evolution process meta-model (EPMM)³⁰ as a formal tool used to define software evolution processes. EPMM not only formally defines the structures and behaviors of all components, eg, tasks, activities, and software processes, but also embodies the important properties of software evolution process, eg, iteration, concurrency, interleaving, feedback-driven, and multi-level, are embodied. Therefore, EPMM can be used to model software processes and software evolution processes at different abstract levels. According to the decomposition granularity of software processes, a four-level framework is designed in EPMM. The highest abstract level is a global level that includes all software processes. The third through first levels are the software process level, activity level, and task level, respectively.

Based on the definition of the TRMM, EPMM is used to model base processes for hard goals. For specific trustworthiness goals, TG_activities must be inserted into the base processes. Therefore, using aspect-oriented modeling to extend the EPMM framework, a TROSP modeling framework is proposed.

3.1 | TROSP modeling framework

Figure 3 depicts the TROSP modeling framework.

At the activity level, TG_activities are added. Based on the definition of TG_activity in Definition 3, an activity body is either a software process or a set of tasks. When a TG_activity is a software process, it is defined as a TG_process aspect. Similarly, when a TG_activity is decomposed into a set of tasks, each task is defined as a TG_task aspect. After weaving these aspects into the base processes, trustworthiness processes are added to the global level.

At the process level, a number of formal components of TG_process aspects are defined. These components are the pointcut, advice, aspect, and weaving mechanism. A pointcut is a well-defined position in the base program where additional behavior can be attached.³⁵ In the modeling domain, a pointcut represents a well-defined element in the base model where an advice can be introduced. In the following definition, a pointcut in a base process is an element of a Petri net.

Definition 5. (Pointcut). Given a base process $p = (C, A; F, M_0)$, a pointcut pc is a condition $pc \in C$, or an activity $pc \in A$, or a flow relation $pc \in F$ of p , $pc \in PC$, $PC \mapsto \{C, A, F\}$.

An advice specifies how to augment or constrain base processes. Each advice is defined with a set of pointcuts that determine the positions of attachment of the advice. Petri nets are used to define the functions of the advices.

Definition 6. (Advice). An advice is a 5-tuple $ad = (C, A; F, A_e, A_x)$, where (1) $(C, A; F)$ is a Petri net in which C is a set of conditions, A is a set of activities, $A \cup C \neq \emptyset$, and F is a set of flow relations, $F \subseteq (C \times A) \cup (A \times C)$; (2) $A_e, A_x \subseteq A$ are the entrance activity set and the exit activity set of the advice, respectively.

The difference between a base process and an advice is that an advice is a Petri net with no marking, which means the activities in an advice are not enabled. The activities are enabled only when they are woven into the base processes.

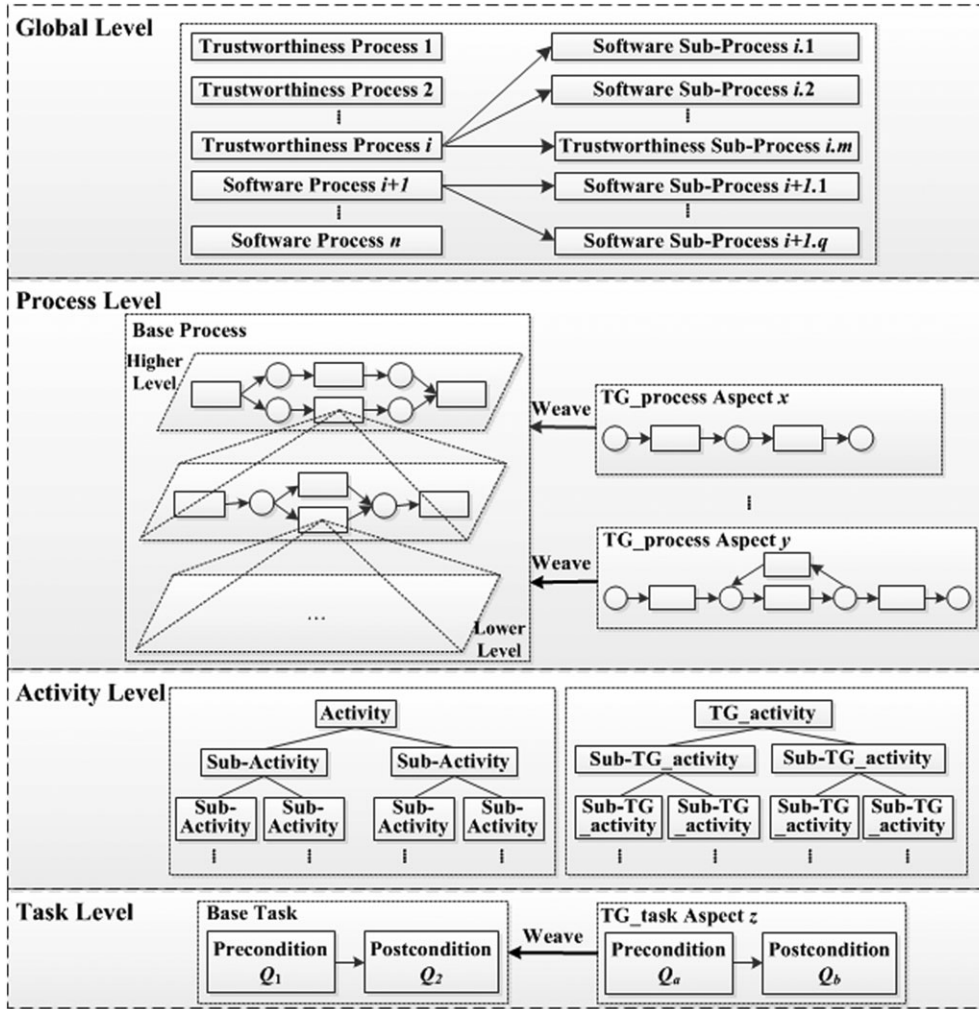


FIGURE 3 TROSP modeling framework

A TG_process aspect is an encapsulated entity of an advice and its pointcuts with weaving types. For the sake of simplicity, the term TG_process aspect is sometimes denoted by aspect in the following.

Definition 7. (TG_Process Aspect). A TG_process aspect is defined by a 2-tuple $tpa = (ad, W)$, where (1) ad is an advice that augments or constrains a base process p ; (2) W is a set of tuples, and $\forall w \in W$ is a 2-tuple $w = (pc, wt)$ in which pc is a pointcut that represents a weaving position in the base process p , $\forall pc \in PC$; wt is a weaving type for the advice ad ; *before*, *after*, *around*, *iteration*, and *concurrency* are five weaving types, and each is described as 1, 2, 3, 4, and 5, respectively.

Similar to the TG_process aspect, a TG_task aspect is an encapsulated entity of a TG_task advice and its TG_task pointcuts with weaving types. The TG_task advice in a TG_task aspect is a 2-assertion that defines the function of the TG_task. The precondition of the 2-assertion defines the state before the TG_task is executed, while the postcondition defines the state after the TG_task is executed. The TG_task advices in the TG_task aspects are also enabled only when they are woven into the base tasks.

The weavings of aspects are specified by a weaving mechanism. Two types of weaving are defined in the weaving mechanism: asymmetric weaving and symmetric weaving. Asymmetric weaving weaves aspects into base processes, whereas symmetric weaving weaves multiple aspects together when these aspects have joint pointcuts with the same weaving types.³⁶ For intuitive analysis, in the following, asymmetric weaving is called base-aspect weaving, and symmetric weaving is called aspect-aspect weaving.

Given a base process p and a set of TG_process aspects $TPA = \{tpa_1, tpa_2, \dots, tpa_j\} (j > 0)$, the weaving begins by creating a weaving plan to store the pointcuts for all these aspects, such as the examples shown in Table 4. In this plan, aspects tpa_1 and tpa_2 have the joint pointcut pc_2 and the same weaving type 1 (*before* type in Definition 7). Aspects tpa_2 and tpa_j have the joint pointcut pc_i with a different weaving type. At pointcut pc_1 , only tpa_1 will be woven with weaving type 5 (*concurrency* type in Definition 7).

When several aspects are woven at joint pointcuts with the same weaving types, such as tpa_1 and tpa_2 at pc_2 , the interdependencies among these aspects are analyzed first. The aspects are then woven together based on the interdependence relations. If there is no interdependence among these aspects, they can be woven in concurrent relations for higher efficiency. Aspects that have joint pointcuts but different weaving

TABLE 4 A weaving plan example

Aspects	Pointcuts			
	pc_1	pc_2	...	pc_i
tpa_1	5	1		
tpa_2		1		2
...				
tpa_j				3

types, such as tpa_2 and tpa_j at pc_i , or that do not have joint pointcuts with the other aspects, such as tpa_1 at pc_1 , are woven into the base processes separately. After all TG_process aspects are woven into the base processes, the woven trustworthiness processes are generated and added to the global model at the global level (see Figure 3).

Definition 8. (Global Model). A global model is a 3-tuple $g = (P, TP, E)$, where (1) P is a set of base processes; (2) TP is a set of trustworthiness processes; and (3) $E \subseteq (P \times P) \cup (TP \times TP) \cup (TP \times P)$ is a binary relation and a partial order, called the embedded relation of P and TP . $E = \{(x, x') \mid x, x' \in TP \cup P \wedge x' \text{ is embedded in } x\}$, and x' is called a subprocess of x .

The aspect-oriented paradigm provides a proper mechanism to weave the TG_process aspects into the base processes. However, changing the base processes via aspect-oriented modeling may introduce mistakes or errors. As mentioned previously, process trustworthiness is the degree of confidence that the software process will produce the expected trustworthy work products. A process that has errors is not trustworthy. To ensure the correctness of the changes to the base processes, a correctness weaving method for aspect-oriented trustworthiness process modeling is proposed.

3.2 | TROSP modeling correctness

Based on the weaving mechanism, two types of correctness are analyzed: aspect-aspect correctness and base-aspect correctness.

3.2.1 | Aspect-aspect correctness

Jointly deployed aspects may interact with each other. If not treated properly, an interaction can give rise to interferences. Aspect-aspect interferences are caused by the interdependencies among aspects. Li³⁰ used Bernstein's sufficient condition for the independence of two programs to define activity dependences. He defined data dependence and control dependence for activities in software processes. Data dependence specifies that the execution of an activity should precede the execution of another activity. Control dependence expresses conditional execution dependencies between activities; that is, it specifies that an activity is conditionally executed depending on the execution result of another activity. Based on these two types of dependences, aspect-aspect correctness is defined.

Definition 9. (Aspect-Aspect Correctness). Let tpa_i and tpa_j be any two aspects; the correctness of aspect-aspect weaving is defined as follows: (1) tpa_i and tpa_j are woven in order of data dependence iff $O(tpa_i) \cap I(tpa_j) \neq \emptyset$ or $O(tpa_j) \cap I(tpa_i) \neq \emptyset$ ($O(tpa)$ and $I(tpa)$ are the set of output artifacts and input artifacts of tpa as defined in Definition 3); (2) tpa_i and tpa_j are woven according to the control dependence iff whether tpa_j can be executed is determined by the execution result of tpa_i .

To analyze and describe the dependence relations intuitively, a dependence graph³⁰ is constructed. For example, assume that reliability and accuracy are two trustworthiness goals. The TG_activities for reliability include *reliability identification*, *reliability modeling*, and *fault tolerance design*. A TG_activity for accuracy is *accuracy boundary identification*. Assume that *reliability identification* and *reliability modeling* are encapsulated in the TG_process aspects tpa_i and tpa_j . They are woven into a joint pointcut with the same weaving type. Since the outputs of *reliability identification* will be inputted into *reliability modeling*, their dependence relation is data dependence. Figure 4A,B depicts the data dependence graph and Petri net transformed from the dependence graph. Because the exit activity set A_x of tpa_i and the entrance activity set A_e of tpa_j are both single activity sets, the transformed Petri net can be simplified, as shown in Figure 4C. Similarly, assume that there is control dependence between *accuracy boundary identification* and *fault tolerance design*. Figure 4D,E,F depicts their control dependence graph, transformed Petri net, and simplified Petri net. The execution of *fault tolerance design* is determined by the execution result of *accuracy boundary identification*. When the result of *accuracy boundary identification* shows that the failure of any part of software is not acceptable, *fault tolerance design* will not be executed and the token will be moved to a_v . a_v in Figure 4F represents a virtual activity that does nothing but transfers tokens.

For tpa_i and tpa_j , ad_i and ad_j are executed sequentially because of their data dependence relation. Assume that the woven aspect is $tpa = (ad, W)$, where $ad = (C, A; F, A_e, A_x)$ and $W = \{(pc, wt)\}$. Formally, $C = ad_i, C \cup ad_j, C, A = ad_i, A \cup ad_j, C \cup \{a_v\}, F = ad_i, F \cup ad_j, F \cup \{(c_s, v) \mid c_s \in ad_1, A_x^*\} \cup \{(v, c_t) \mid c_t \in ad_2, *A_e\}, A_e = ad_i, A_e, A_x = ad_j, A_x, pc = pc_i \cap pc_j, wt = wt_i, A_x^*$ and $*A_e$ denote the output set of A_x and the input set of A_e , respectively (let $p = (C, A; F, M_0)$ be a software process and $x \in C \cup A$, then $*x = \{y \mid y \in C \cup A \wedge (y, x) \in F\}$ and $x^* = \{y \mid y \in C \cup A \wedge (x, y) \in F\}$). For tpa_f and tpa_i , the execution of ad_i of tpa_i is determined by ad_f of tpa_f because of their control dependence relation.

The weaving between TG_task aspects is similar and omitted for the sake of simplicity.

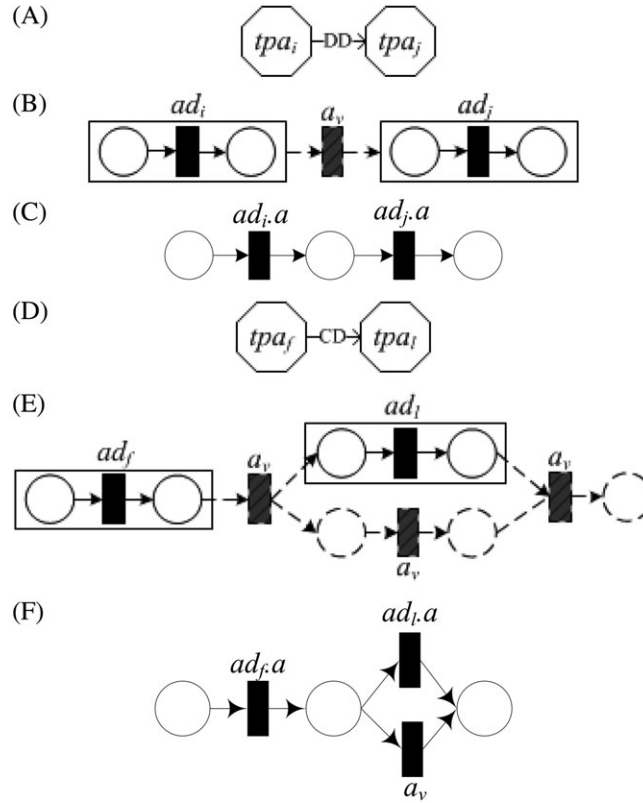


FIGURE 4 Examples of dependence graphs and transformed petri nets: A, data dependence graph B, transformed petri net C, simplify transformed petri net D, control dependence graph E, transformed petri net F, simplify transformed petri net

3.2.2 | Base-aspect correctness

The base-aspect weaving weaves aspects into the base processes. According to the properties of Petri nets, when an aspect is woven into a base process, the errors or mistakes that may occur can be defined as the structural or dynamic property problems of Petri nets. When an aspect is woven into a base process, if structural property problems arise, there must be static structural flaws existing in the woven model. Referring to the structural properties of Petri nets, the correct structural properties of an aspect-oriented trustworthiness process are no side condition, no isolated node, no deadlock, and no trap. When an aspect-oriented trustworthiness process is executed, its dynamic properties should be discussed. The dynamic properties are relevant to the marking of the process model (the marking is defined in Definition 1). Similarly, referring to the dynamic properties of Petri nets, the correct dynamic properties are safe, contact-free, persistent, and liveness.

When an aspect is woven into a base process, errors or mistakes only affect the activities around the pointcuts. For example, an aspect $tpa = (\{c_{tpa1}, c_{tpa2}\}, \{a_{tpa1}\}; \{c_{tpa1}, a_{tpa1}\}, (a_{tpa1}, c_{tpa2}\}), (p, a_{p2}, 3))$ is woven into a base process $p = (\{c_{p1}, c_{p2}, c_{p3}\}, \{a_{p1}, a_{p2}\}; \{c_{p1}, a_{p1}\}, (a_{p1}, c_{p2}), c_{p2}, a_{p2}, (a_{p2}, c_{p3}), \{c_{p1}\})$. The affected activities are a_{tpa1} and a_{p2} . Only these affected activities and the connected conditions should be analyzed. We define them in a weaving region. A weaving region is a Petri net that is part of a woven trustworthiness process. In Figure 5, $wr = (\{c_{p2}, c_{p3}\}, \{a_{tpa1}, a_{p2}\}; \{c_{p2}, a_{p2}\}, (a_{p2}, c_{p3}), (c_{p2}, a_{tpa1}), (a_{tpa1}, c_{p3}))$ is the weaving region that is enclosed in a dashed rectangle.

Definition 10. (Weaving Region) Let $tpa = (ad, W)$ be a TG_process aspect and $p = (C, A; F, M_0)$ be a base process. Suppose that tpa is woven into p at pointcut $pc \in PC$; a weaving region is $wr = (C', A'; F')$, where, $C' = \{c | c \in \bullet A' \cup A' \bullet \wedge c \in p. C \cup ad. C\}$, $A' = \{a | a \in \bullet PC \cup PC \bullet \cup PC \cup \text{dom}(PC) \cup \text{cod}(PC) \wedge a \in p. A \cup ad. A\} \cup ad. A_e \cup ad. A_x$, $F' = \{f | f \in (C' \times A') \cup (A' \times C')\}$.

Assume that all base processes and aspects have been proved to be satisfied with respect to structural and dynamic properties. The correctness analysis is only focused on the weaving region. In the following, structural correctness and dynamic correctness are defined. Structural correctness is determined by the topology structure properties of Petri nets. Dynamic correctness is relevant to the initial marking of Petri nets.

Definition 11. (Structural Correctness) Given a TG_process aspect $tpa = (ad, W)$ and a base process $p = (C, A; F, M_0)$, when tpa is woven into p , the structural correctness of the weaving region wr is defined as follows: (1) if $\forall a \in wr. A$, then $\bullet a \cap a \bullet = \emptyset$; (2) if $\forall y \in wr. C \cup wr. A$, then $\bullet y \cup y \bullet \neq \emptyset$; (3) if $C_1 \subseteq wr. C$, there is no $\bullet C_1 \subseteq C_1 \bullet$ or $C_1 \bullet \subseteq \bullet C_1$.

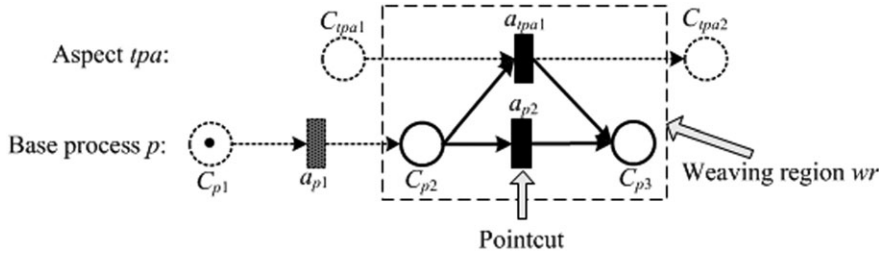


FIGURE 5 An example of a weaving region

In Definition 11, the first structural correctness indicates that an activity in the weaving region does not have side condition. If an activity has a side condition, it will never fire. The second structural correctness shows that isolated nodes are not allowed. The third structural correctness implies that the conditions in the weaving region do not have deadlock conditions ($\bullet C_1 \subseteq C_1^*$) or trap conditions ($C_1^* \subseteq \bullet C_1$). A deadlock condition without a token will never be able to obtain a token. A trap condition with a token will never be able to send out the token.

Definition 12. (Dynamic Correctness) Given a base process $p = (C, A; F, M_0)$ and a TG_process aspect $tpa = (ad, W)$, when tpa is woven into p , the dynamic correctness of the weaving region wr is defined as follows: (1) if $\forall a \in ad. A$ and $\exists M \in \Sigma(p, M_0)$, then $M[a >$, where $\Sigma(p, M_0)$ is a set that includes markings reachable from the initial marking M_0 ; (2) if $\forall a \in wr. A \cap p. A$, $\exists \sigma \in ad. A^*$, and $\exists M \in \Sigma(p, M_0)$, $M[a > \vee ((\neg M[a > \wedge M[\sigma > M'] \rightarrow M'[a >)$, where σ is an activity subset of the powerset of activities in ad ; (3) if $\forall c \in wr. C$ and $\forall M \in \Sigma(p, M_0)$, then $M(c) \leq 1$; (4) if $\forall a \in wr. A \forall M \in \Sigma(p, M_0)$, $\exists c \in \bullet a$, and $M(c) = 1$, there is no $\exists c' \in a^*$ and $M(c') = 1$.

In this definition, the first dynamic correctness implies that there exists a reachable marking M and an activity in the aspects can be enabled at M . The second dynamic correctness shows that each activity in the base process is also enabled. Because the software processes in this paper are defined in the elementary Petri net, conditions that are not safe or contact are not allowed. Thus, the third dynamic correctness indicates that all conditions in the weaving region are safe. The fourth dynamic correctness shows that all conditions in the weaving region are contact-free. The safe and contact-free conditions ensure that the activities after these conditions can be enabled.

When an aspect is woven into a base process, there are 10 types of base-aspect weaving operations: *before condition weaving*, *after condition weaving*, *around condition weaving*, *iteration weaving*, *concurrency weaving*, *before activity weaving*, *after activity weaving*, *around activity weaving*, *condition-activity flow weaving*, and *activity-condition flow weaving*. Based on the correctness Definitions 11 and 12, all of these weaving operations are analyzed, and four of them do not satisfy the correctness definitions. Here, we assume that an aspect $tpa = (ad, W)$ is woven into a base process $p = (C, A; F, M_0)$.

Before activity weaving weaves the aspect tpa before an activity pointcut a of base process p , and the weaving type is *before weaving* ($w = (p, a, 1)$), as shown in Figure 6A. The original intention of this weaving is to add a restriction to the activity a in the base process p that the execution of a must wait for the execution of the aspect tpa . Therefore, a new input condition c is added to the activity a . However, when the activity a executes repeatedly in an iterative structure, this weaving will forbid its iterative execution because there will be no token in the new input condition c after the activity a executes. This violates the second dynamic correctness in Definition 12. Therefore, we replace *before activity weaving* with *condition-activity flow weaving* because they have the same effect on weaving. Figure 6B and Algorithm 2 depict and describe *condition-activity flow weaving*, respectively.

Algorithm 2 Flow_Weaving.

INPUT: $p = (C, A; F, M_0)$, $tpa = (ad, W)$.

OUTPUT: $tp = (C', A'; F', M_0')$.

BEGIN

$C' := \emptyset$; $A' := \emptyset$; $F' := \emptyset$; $M_0' := \emptyset$;

FOR each w in W DO

IF $\text{dom}(w.pc) \in p.C$ THEN /*condition-activity flow weaving*/

BEGIN

$C' := p.C \cup ad.C - ad.\bullet A_e$;

IF $|A_e| > 1$ THEN

BEGIN

$A' := p.A \cup ad.A \cup \{a_v\}$;

$F' := p.F \cup ad.F - w.pc \cup \{(a_v, c_i) \mid c_i \in ad.\bullet A_e\} \cup \{(\text{dom}(w.pc), a_v)\} \cup \{(c_j, \text{cod}(w.pc) \mid c_j \in ad.A_x^*)\}$

/* a_v represents a virtual activity that does nothing but transfers tokens.*/

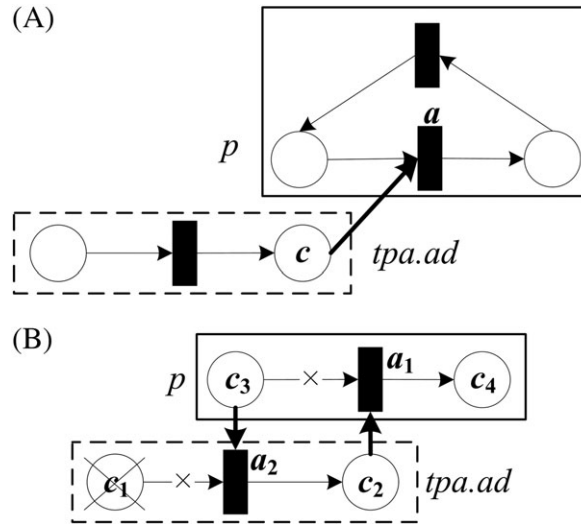


FIGURE 6 Before activity weaving is replaced by condition-activity flow weaving: A, before activity weaving; B, condition-activity flow

```

END
ELSE
BEGIN
 $A' := p. A \cup ad. A;$ 
 $F' := p. F \cup ad. F - w. pc - inflow(ad. A_e) \cup \{(dom(w. pc), a_i) \mid a_i \in ad. A_e\} \cup \{(c_j, cod(w. pc)) \mid c_j \in ad. A_x^*\}$ 
END;
 $M_0' := p. M_0$ 
END
END

```

When using Algorithm 2 in a *condition-activity flow weaving*, Figure 6B shows the result of weaving an aspect tpa into a base process p . The pointcut of tpa is a flow relation $w.pc = (c_3, a_1)$. Following the procedure of Algorithm 2, since $A_e = \{a_2\}$, we can get $ad. A_e = c_1$ and $C' = \{c_2, c_3, c_4\}$. Similarly, because $|A_e| = 1$, we can get $A' = \{a_1, a_2\}$ and $F' = \{(a_1, c_4), (a_2, c_2), (c_3, a_2), (c_2, a_1)\}$, where $dom(w.pc) = c_3$, $cod(w.pc) = a_1$ ($dom()$ and $cod()$ are defined in Definition 2), and $inflow(ad. A_e) = \{(c, a) \mid (c, a) \in ad. F, c \in ad. C \wedge a \in ad. A_e\} = \{(c_1, a_2)\}$. C' , A' , and F' are the conditions set, activities set, and flow relations set of the output trustworthiness process tp .

After activity weaving is similar to *before activity weaving* but has the opposite weaving type. As shown in Figure 7A, $w = (p. a, 2)$, the pointcut is activity a in base process p , and the weaving type is *after weaving*. This weaving may cause a contact problem (the fourth dynamic correctness in Definition 12) when a is in an iterative structure. Therefore, it is replaced by the *activity-condition flow weaving* (see Figure 7B).

Before condition weaving weaves an aspect tpa before a condition pointcut c of base process p , and the weaving type is *before weaving* ($w = (p. c, 1)$). As shown in Figure 8A, the result of this weaving is to add the advice ad before the condition c . Nevertheless, based on Definition 12, this weaving may cause a safety problem (the third dynamic correctness in Definition 12) in c or make the activities in the aspect unable to fire (the first dynamic correctness in Definition 12). *After condition weaving* is similar to *before condition weaving* but has the opposite weaving type, as shown in Figure 8B. This weaving may cause the persistent problem (the second dynamic correctness in Definition 12) in the base process. These two weavings cannot be replaced because the relations between activities in the aspect and base process are not specified.

Similarly, the other six weaving operations are analyzed and proved to be correct. These operations are *condition-activity flow weaving*, *activity-condition flow weaving*, *around condition weaving*, *around activity weaving*, *iteration weaving*, and *concurrency weaving*. Therefore, these six weavings are used to weave aspects into base processes.

When weaving TG_task aspects into base tasks, because no message is transferred, there is no correctness problem.

3.3 | TROSP modeling workflow and aided tool

The approach to modeling TROSP is linked to the TRMM (see Section 2) and the TROSP modeling framework (see Section 3.1). The steps of the proposed modeling approach are depicted in Figure 9. The first step is the negotiation among the multiple stakeholders to reconcile their trustworthiness goals and soft goals. The main method is the Delphi method,²¹ which is a structured communication technique based on the results of questionnaires sent to a panel of experts. The second and third steps model the TRM and find satisfied TG_activities by backward reasoning.

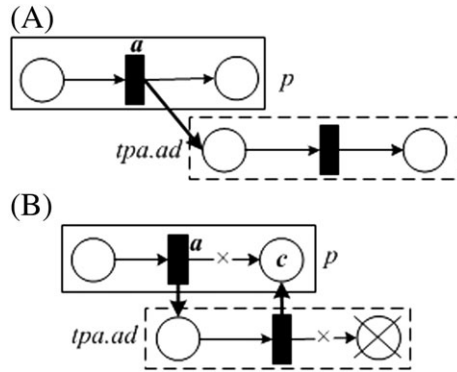


FIGURE 7 After activity weaving is replaced by activity-condition flow weaving: A, after activity weaving; B, activity-condition flow weaving.

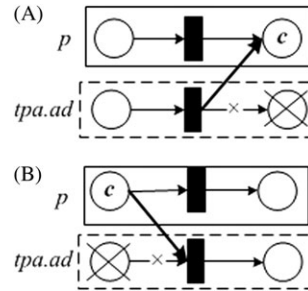


FIGURE 8 Before condition weaving and after condition weaving: A, before condition weaving; B, after condition weaving

For each TG_activity, the input artifacts set I , local artifacts set L , output artifacts set O , and analyzing activity body B are defined. If B is a software process, a corresponding TG_process aspect is defined. Otherwise, B is decomposed into a set of tasks, and a set of corresponding TG_task aspects are defined. Then, abiding by the definitions of Aspect-Aspect Correctness (see Definition 9), Structural Correctness (see Definition 11), and Dynamic Correctness (see Definition 12), these aspects are woven at the process level and the task level. After all aspects are woven, the global model is generated. Algorithm 3 describes the modeling at the process level and the global level. The output TP and g are a set of trustworthiness processes and a global model, which are defined in Definition 8.

Algorithm 3 Process-Global_Modeling.

INPUT: $P = \{p_i\}_{0 < i < m}$, $TPA = \{tpa_j\}_{0 < j < k}$.

OUTPUT: TP , g .

BEGIN

$TP := \emptyset$;

FOR $\exists p \in P$ AND $\exists tpa. w. pc \in p. C \cup p. A \cup p. F$ DO

BEGIN

$PC := tpa_1. w. pc \cup tpa_2. w. pc \cup \dots \cup tpa_k. w. pc$;

FOR $\forall pc \in PC$ DO

BEGIN

$SJP_{pc} := \emptyset$;

FOR $j = 1$ to k DO

IF $pc \in tpa_j. w. pc$ THEN

$SJP_{pc} := SJP_{pc} \cup \{tpa_j\}$

END;

FOR $\forall pc \in PC$ DO

BEGIN

FOR $|SJP_{pc}| > 1$ and tpa with same wt DO /*Aspect-Aspect_Weaving*/

BEGIN

$tpa := \text{Aspect - Aspect_Weaving}()$;

$SJP_{pc} := SJP_{pc} \cup \{tpa\}$

END;

END;

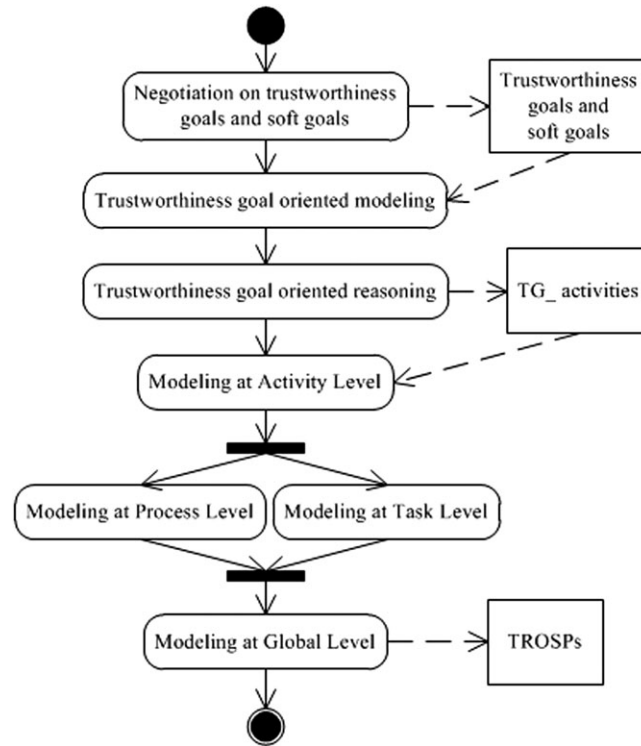


FIGURE 9 TROSP modeling workflow

```

END;
FOR |SJPpc| = 1 or tpa with different wt DO /*Base-Aspect_Weaving*/
  IF tpa. w. pc ∈ p. A THEN
    tp := Activity_Weaving (p, tpa);
  ELSE IF tpa. w. pc ∈ p. C THEN
    tp := Condition_Weaving (p, tpa);
  ELSE IF tpa. w. pc ∈ p. F THEN
    tp := Flow_Weaving (p, tpa);
  END;
  TP := TP ∪ {tp};
  Replace p with tp in E;
  P := P - {p}
END
END.

```

In Algorithm 3, SJP is a set of aspects. The aspects in SJP have joint pointcuts. For a specific pointcut pc , the aspects in SJP_{pc} have the joint pointcut pc . When several aspects are woven at joint pointcuts, interdependencies among these aspects should be analyzed and the aspects with the same weaving types are woven together first. Then, these aspects are woven into the base processes. When a base process p is woven with all the aspects, it becomes a woven trustworthiness process tp , and p in E is replaced by tp . E is a set of embedded relations of base processes set P and trustworthiness processes set TP in Definition 8. The proofs of the correct modeling in Algorithm 3 will be presented in Sections 5.1.2 and 5.1.3 below.

With the aim of TROSP modeling, the Trustworthiness Process Aided Tool (TPAT) modeling tool is designed and developed in the open-source software PIPE³⁷ with a plug-in technique. TPAT consists of two core components: the base processes modeling component PIPE and the aspect-oriented extension component, as illustrated in Figure 10. The aspect-oriented extension component is constrained by the definitions of the base processes and the aspects and consists of two weaving components: aspect-aspect weaving and base-aspect weaving. All definitions and weaving data are written in a configuration file.

To model a TROSP, TPAT functions as follows.

- (1) The base processes and the advices of the aspects are first created in PIPE.

- (2) According to the definitions of the base processes and the aspects, a weaving plan is created and stored in a configuration file. This weaving plan is created for the initialization of the aspect-oriented extension, as introduced in the weaving mechanism in Section 3.1. The aspects weavings are enforced by clicking the plug-in TPAT in PIPE.
- (3) Aspect-aspect correctness is followed to weave together the aspects that are woven at the joint pointcuts with the same weaving types. The remaining aspects are woven into the base processes at each of the pointcuts according to the six base-aspect weaving operations. The correctness of the weaving is examined for the whole weaving procedure. Any correctness problem will terminate the weaving procedure, and messages that state the error positions will pop up. Once all errors are removed, the procedure will continue until the TROSPs are generated.

4 | CASE STUDY

The proposed method was applied to a systematic study of security infrastructure system (SIS) software. SIS is a trustworthy third-party certification authority software system. It provides identity authentication services and secure connections over the Internet. SIS certification authority (SISCA) and SIS user agent (SISUA) are two subsystems of SIS. SISCA is a server system that manages users, keys, certificates, and cross authorization. SISUA is a client system that helps users encrypt, decrypt, sign, and verify. SIS is used to issue and manage digital certificates for identity authentication. The certificate owners are verified by the signature in the digital certificates. Keys are used for secure data transfer. Over the years, new evolution requirements for running SIS software have been proposed continually.

With the aim of evolution of the SIS software, a project team was formed involving the corresponding stakeholders. The team comprised a software engineering technical manager, a superintendent of the SIS, an expert in Public Key Infrastructure, a deputy of the certificate owner, a software developer and a maintenance team.

4.1 | Trustworthiness goal-oriented modeling and reasoning

The original functional requirements and NFRs were summarized from the software requirement specifications and provided to the project team. Through the negotiation in the Delphi procedure, new TRs were acquired. Table 5 lists the trustworthiness goals and the soft goals.

To satisfy the trustworthiness goals, TG_activities were chosen to model the TRM, as shown in Figure 11.

The corresponding Boolean formula ϕ is the following:

$$\phi ::= \phi_{Initial} \wedge \phi_{Model} \wedge \phi_{Constraint} \wedge \phi_{Conflict}$$

In ϕ , the Boolean formula of the initial values $\phi_{Initial}$ that are assigned to the trustworthiness goals and the soft goals is the following:

$$\phi_{Initial} ::= \boxed{SA(TG_1)} \wedge \boxed{SA(TG_2)} \wedge PS(TG_3) \wedge SA(TG_4) \wedge PS(TG_5) \wedge PS(SG_1) \wedge PS(SG_2)$$

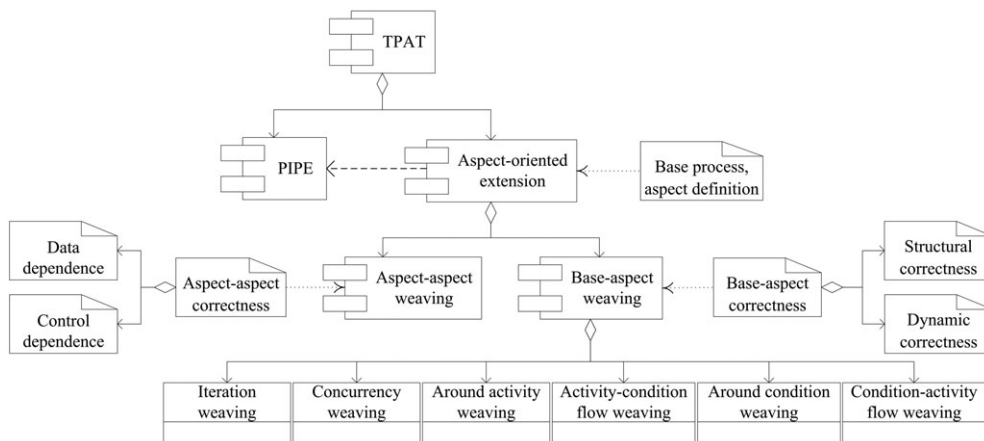


FIGURE 10 TPAT component design

The Boolean formula of the TRM ϕ_{Model} is the following:

$$\begin{aligned} \phi_{Model} ::= & (\neg SA(TG_1) \vee SA(TG_{11})) \wedge (\neg SA(TG_{11}) \vee SA(ta_{111})) \wedge (\neg SA(TG_{11}) \vee SA(ta_{112})) \wedge (\neg SA(TG_{11}) \vee SA(ta_{113})) \wedge (\neg SA(TG_1) \vee SA(TG_{12})) \\ & \wedge (\neg SA(TG_{12}) \vee SA(ta_{121})) \wedge (\neg SA(TG_{12}) \vee SA(ta_{122})) \wedge (\neg PS(TG_5) \vee PS(ta_{51})) \wedge (\neg PS(TG_5) \vee PS(ta_{52})) \wedge (\neg PS(TG_5) \vee PS(ta_{53})) \\ & \wedge (\neg SA(TG_4) \vee SA(ta_{41})) \wedge (\neg SA(TG_4) \vee SA(ta_{44})) \wedge (\neg SA(TG_4) \vee SA(ta_{42})) \wedge (\neg SA(ta_{42}) \vee SA(ta_{421})) \wedge (\neg SA(ta_{42}) \vee SA(ta_{422})) \\ & \wedge (\neg SA(ta_{42}) \vee SA(ta_{423})) \wedge (\neg SA(ta_{42}) \vee SA(ta_{424})) \wedge (\neg SA(TG_4) \vee SA(ta_{43})) \wedge (\neg SA(ta_{43}) \vee SA(ta_{431})) \wedge (\neg SA(ta_{43}) \vee SA(ta_{432})) \\ & \wedge (\neg SA(ta_{43}) \vee SA(ta_{433})) \wedge (\neg SA(TG_4) \vee SA(TG_{41})) \wedge (\neg SA(TG_{41}) \vee SA(ta_{411})) \wedge (\neg SA(TG_{41}) \vee SA(ta_{412})) \wedge (\neg SA(TG_4) \vee SA(TG_{42})) \\ & \wedge (\neg SA(TG_{42}) \vee SA(ta_{425})) \wedge (SA(TG_{42}) \vee SA(ta_{426})) \wedge (\neg SA(TG_{42}) \vee SA(ta_{427})) \wedge (\neg SA(TG_4) \vee SA(TG_{43})) \wedge (\neg SA(TG_{43}) \vee SA(ta_{434})) \\ & \wedge (\neg SA(TG_{43}) \vee SA(ta_{435})) \wedge (SA(TG_2) \vee SA(ta_{21})) \wedge (\neg SA(TG_{21}) \vee SA(ta_{211})) \wedge (\neg SA(TG_2) \vee SA(ta_{21})) \wedge (\neg SA(TG_2) \vee SA(ta_{22})) \\ & \wedge (\neg SA(ta_{22}) \vee SA(ta_{221})) \wedge (\neg SA(ta_{22}) \vee SA(ta_{222})) \wedge (\neg SA(ta_{22}) \vee SA(ta_{223})) \wedge (\neg SA(ta_{22}) \vee SA(ta_{224})) \wedge (\neg SA(TG_2) \vee SA(ta_{23})) \\ & \wedge (\neg SA(ta_{23}) \vee SA(ta_{231})) \wedge (\neg SA(ta_{23}) \vee SA(ta_{232})) \wedge (\neg SA(ta_{23}) \vee SA(ta_{233})) \wedge (\neg SA(ta_{23}) \vee SA(ta_{234})) \wedge (\neg SA(TG_2) \vee SA(ta_{24})) \\ & \wedge (\neg SA(ta_{24}) \vee SA(ta_{241})) \wedge (\neg SA(TG_2) \vee SA(ta_{25})) \wedge (\neg PS(TG_3) \vee PS(ta_{31})) \wedge (\neg PS(ta_{31}) \vee PS(ta_{311})) \wedge (\neg PS(ta_{31}) \vee PS(ta_{312})) \\ & \wedge (\neg PS(TG_3) \vee PS(ta_{32})) \wedge (\neg SA(TG_1) \vee DE(ta_{427})) \wedge (\neg SA(TG_3) \vee DE(ta_{435})) \wedge (\neg SA(TG_2) \vee DE(ta_{427})) \wedge (\neg SA(TG_2) \vee DE(ta_{51})) \\ & \wedge (\neg SA(TG_3) \vee DE(ta_{435})) \wedge (\neg SA(TG_3) \vee DE(ta_{21})) \wedge (\neg SA(TG_3) \vee DE(ta_{51})) \wedge (\neg PS(SG_1) \vee DE(ta_{21})) \wedge (\neg PS(SG_1) \vee DE(ta_{211})) \\ & \wedge (\neg PS(SG_1) \vee DE(ta_{435})) \wedge (\neg PS(SG_2) \vee DE(ta_{21})) \wedge (\neg PS(SG_2) \vee DE(ta_{211})) \end{aligned}$$

The Boolean formula of the constraints $\phi_{Constraint}$ is the following:

$$\phi_{Constraint} ::= (SA(ta_{425}) \vee SA(ta_{427})) \wedge (SA(ta_{426}) \vee SA(ta_{427}))$$

The Boolean formula of the conflict $\phi_{Conflict}$ that should be avoided is *strong conflict*:

$$\phi_{Conflict} ::= \neg (SA(n) \wedge DE(n)), \quad n \text{ is the node in } m$$

By using backward reasoning in the modified zChaff, the result for the first reasoning was “UNSAT,” and the conflict was at the 28th clause $SA(ta_{42}) \rightarrow SA(ta_{427})$, as shown in Figure 12A. *Fault Tolerance Design* (ta_{427}) conflicts with the initial satisfaction values of TG_1 and TG_2 in $\phi_{Initial}$, as shown in the Boolean formula of the TRM ϕ_{Model} and the Boolean formula of the initial values $\phi_{Initial}$. After saving $SAT_Status(ta_{427})$, we deleted this conflicting node ta_{427} in ϕ to continue the reasoning. Then, the results showed that the conflicts were on the other three nodes, ta_{435} , ta_{21} , and ta_{211} . These nodes are *Redundancy Design* activity, *Define Minimum Security Criteria* activity, and *Define Minimum Cryptographic Design*. After similar saving and deleting, the final result was “SAT,” as shown in Figure 12B.

Table 6 lists the resulting statuses and conflicts for all $TG_activities$. After discussion within the project team, *Fault Tolerance Design* was removed and compensated by *Fault Prevention Design* and *Error Correction Design*. The other four conflicting $TG_activities$ were analyzed during prototyping to make appropriate design decisions. Additionally, when the four conflicting $TG_activities$ were woven into the base processes and base tasks, interdependences and weaving correctness were used to control their conflicts. The final modeling of these $TG_activities$ at the activity level, including decomposition and defining aspects, is listed in the rightmost column of Table 6.

4.2 | Trustworthiness requirement-oriented software process modeling

After the preceding reasoning, 44 $TG_activities$ were defined at the activity level. Through thoughtful analysis and team discussions, 18 $TG_activities$ were decomposed into $TG_processes$, and the other 26 $TG_activities$ were decomposed into a set of TG_tasks . The decomposition was based on the purpose of each $TG_activity$ and the base processes that the $TG_activity$ should be integrated in. *Code Review* and *Redundancy Design* are the two $TG_activities$ that we describe in the following as two examples of the 44 $TG_activities$. *Code Review* was decomposed into a $TG_process$. *Redundancy Design* was decomposed into a set of TG_tasks : *Recovery Design*, *N Redundancy Design*, and *Defense Design*.

TG_activity Code Review = (I, O, L, B).

I = {Evolution requests, Code};

O = {Review Report};

TABLE 5 Trustworthiness goals and soft goals for SIS software

Trustworthiness Goals	Soft Goals
TG_1 functional suitability	TG_4 reliability
TG_{11} functional completeness	TG_{41} availability
TG_{12} functional correctness	TG_{42} fault-tolerance
TG_2 security	TG_{43} recoverability
TG_{21} confidentiality	TG_5 compatibility
TG_3 maintainability	
	SG_1 performance
	SG_2 usability

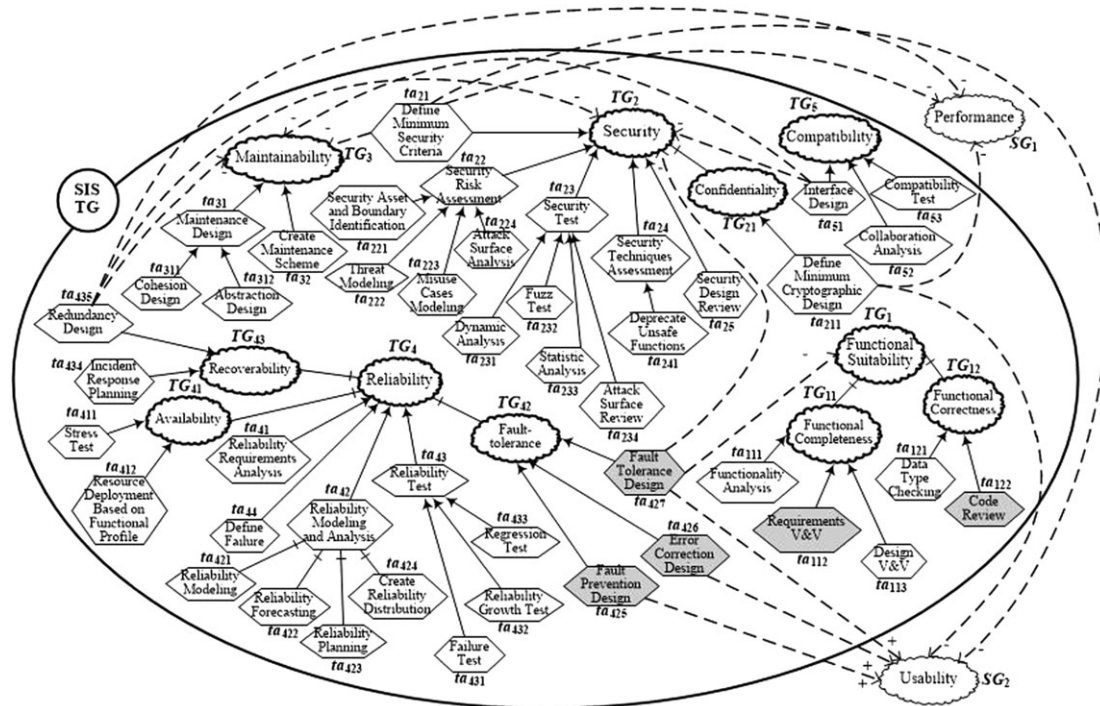


FIGURE 11 TRM for SIS

$L = \{Code\ Review\ Tools\};$

$B = \{Code\ Review\};$

TG_activity Redundancy Design = $((I, O, L, B).$

$I = \{Requirements, Reliability\ Model\};$

$O = \{Design\ Redundancy\};$

$L = \{Design\};$

$B = \{Recovery\ Design, N\ Redundancy\ Design, Defense\ Design\};$

By making use of Li's approach³⁰ at the process level, a series of software processes were generated, and their definitions were checked before weaving aspects into them. For the sake of simplicity, the SIS process is provided as one example process in this paper. The *Design Evolution* activity in the SIS process was decomposed into another example process, the Design Evolution process. In Figures 13 and 14, Petri nets outside the dotted boxes are these two base processes, respectively.

To integrate the TG_activities into the SIS process and Design Evolution process, the decomposed TG_processes and TG_tasks were encapsulated in aspects. *Code Review* is an example that is encapsulated in the TG_process aspect CR.

TG_process aspect_{CR} = (ad, W)

$ad = (C, A, F, A_e, A_x);$

$C = \{c_1', c_2'\};$

$A = \{Code\ Review\};$

$F = \{(c_1', Code\ Review), (Code\ Review, c_2')\};$

$A_e = \{Code\ Review\};$

$A_x = \{Code\ Review\};$

$W = \{w\}, w = (pc, wt);$

$pc = \{(Evolution\ Test, c_5)\};$

$wt = 2; /*activity-condition\ flow\ weaving*/.$

Since the weaving of the TG_process aspects changes the base processes, which may introduce mistakes or errors, the correctness of their weavings was checked. First, the interdependencies among the aspects were analyzed, and the aspects with joint pointcuts and the same weaving types were woven together based on their data dependencies and control dependencies. Then, the woven aspects and all other aspects were woven into the base processes. During the base-aspect weaving procedure, the correctness problems of side condition, isolated node, deadlock, trap, safety, contact-free, and liveness were checked. When correctness problems were found, warning messages popped up in TPAT, and the weaving procedure was terminated until the problems were solved. The final weaving results are illustrated in Figures 13 and 14. Petri nets in the dotted boxes are the TG_process aspects.

(A)

```

root@wangfan-virtual-machine: /home/wangfan/wf/zchaff64
root@wangfan-virtual-machine:/home/wangfan/wf/zchaff64# ./zchaff tennis.cnf
Z-Chaff Version: zChaff 2007.3.12
Solving tennis.cnf .....
conflict at 28
CONFLICT during preprocess

Instance Unsatisfiable
Random Seed Used                0
Max Decision Level              0
Num. of Decisions               0
( Stack + Vsids + Shrinking Decisions ) 0 + 0 + 0
Original Num Variables          65
Original Num Clauses            78
Original Num Literals           149
Added Conflict Clauses          0
Num of Shrinkings               0
Deleted Conflict Clauses        0
Deleted Clauses                 0
Added Conflict Literals         0
Deleted (Total) Literals        0
Number of Implication           63
Total Run Time                  0
RESULT: UNSAT
root@wangfan-virtual-machine:/home/wangfan/wf/zchaff64#
    
```

(B)

```

root@wangfan-virtual-machine: /home/wangfan/wf/zchaff64
Z-Chaff Version: zChaff 2007.3.12
Solving tennis.cnf .....

c 74 Clauses are true, Verify Solution successful.
Instance Satisfiable
1 2 3 4 5 6 7 8 9 10 -11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 3
0 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
57 58 59 60 -61 62 63 64 65 Random Seed Used                0
Max Decision Level                0
Num. of Decisions                  1
( Stack + Vsids + Shrinking Decisions ) 0 + 0 + 0
Original Num Variables             65
Original Num Clauses               74
Original Num Literals              141
Added Conflict Clauses             0
Num of Shrinkings                  0
Deleted Conflict Clauses            0
Deleted Clauses                    0
Added Conflict Literals             0
Deleted (Total) Literals            0
Number of Implication              65
Total Run Time                     0
RESULT: SAT
root@wangfan-virtual-machine:/home/wangfan/wf/zchaff64#
    
```

FIGURE 12 Trustworthiness goal-oriented reasoning for SIS: A, first reasoning result; B, final reasoning result

TABLE 6 Modeling at the activity level

TG_Activity	Status	Conflict	Pointcut and Weaving Type
Functionality analysis	SA		(<i>c</i> ₁ , <i>Techniques Selection</i>), before
Requirements V&V	SA		(<i>Proposal for changes</i> , <i>c</i> ₉), after
Design V&V	SA		<i>Validation</i> , around
Data type checking	SA		Task of <i>Evolution Test</i>
Code review	SA		(<i>Evolution Test</i> , <i>c</i> ₅), after
Interface design	PS/DE	Medium conflict	Task of <i>Re-Design</i>
Collaboration analysis	SA		(<i>c</i> ₁ , <i>Techniques Selection</i>), before
Compatibility test	SA		Task of <i>Evolution Test</i>
Redundancy design	SA/DE	Maintainability, security, performance	Task of <i>Re-Design</i>
Incident response planning	SA		(<i>Integration</i> , <i>c</i> ₈), after
Stress test	SA		Task of <i>Evolution Test</i>
Resource deployment based on functional profile	SA		(<i>c</i> ₃ , <i>Evolution Division</i>), before
Reliability requirements analysis	SA		(<i>c</i> ₁ , <i>Techniques Selection</i>), before
Define failure	SA		(<i>Risk Analysis</i> , <i>c</i> ₁₀), after

(Continues)

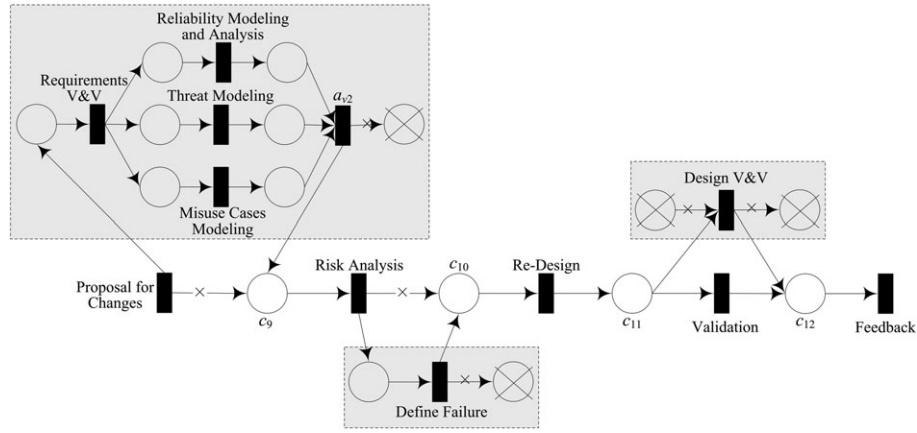


FIGURE 14 Trustworthiness requirement-oriented design evolution process

5 | EVALUATIONS

The following evaluations include analyses of the correctness, performance, and effectiveness of the approach from both theoretical and practical points of view.

5.1 | Formal proofs of correct modeling and reasoning

5.1.1 | Soundness and completeness of the TRM reasoning

The TRM reasoning axiomatization would be soundness if the propagation through the backward axioms was shown to reflect the intended relations of the propagation through the relation axioms, given the same inputs and the constraints on the TRM in Section 2. The completeness of the reasoning shows that we have considered propagation rules for every combination of status label and relation type, given the constraints on the TRM in Section 2.

Let $trm = (N, R)$ be a TRM. Let $n_{x1}, \dots, n_{xi} \in N$ be the TG_activity nodes and $SL(n_{x1}), \dots, SL(n_{xi})$ their status labels. Let $n_{y1}, \dots, n_{yj} \in N$ be the trustworthiness goal nodes and soft goal nodes and $SL(n_{y1}), \dots, SL(n_{yj})$ be their status labels. The following theorems state the soundness and completeness with respect to the axiomatization.

Theorem 1. (Soundness of reasoning) *If there exists a truth value assignment satisfying the relation axioms (RA_1 - RA_{12}), the backward propagation axioms (BA_1 - BA_{14}), and the values $SL(n_{x1}), \dots, SL(n_{xi})$ and $SL(n_{y1}), \dots, SL(n_{yj})$, then $SL(n_{y1}), \dots, SL(n_{yj})$ can be inferred from $SL(n_{x1}), \dots, SL(n_{xi})$ by means of the relation axioms RA_1 - RA_{12} .*

Theorem 2. (Completeness of reasoning) *If $SL(n_{y1}), \dots, SL(n_{yj})$ can be inferred from $SL(n_{x1}), \dots, SL(n_{xi})$ by means of the relation axioms RA_1 - RA_{12} , then there exists a truth value assignment satisfying the relation axioms (RA_1 - RA_{12}), the backward propagation axioms (BA_1 - BA_{14}), and the values $SL(n_{x1}), \dots, SL(n_{xi})$ and $SL(n_{y1}), \dots, SL(n_{yj})$.*

The proofs of the soundness and completeness are similar to the proofs of Sebastiani et al³² and are omitted for the sake of simplicity.

5.1.2 | Correctness of the base-aspect weaving

The structural correctness and dynamic correctness of base-aspect weaving are defined in Section 3.2.2. Let $tpa = (ad, W)$ be a TG_process aspect, $ad = (C, A; F, A_e, A_x)$ be the advice of tpa , and $p = (C, A; F, M_0)$ be a base process. The following theorem states that the advices in the aspects are enabled when they are woven into the base processes. The other theorems and proofs of structural correctness and dynamic correctness are similar and are omitted for the sake of simplicity.

Theorem 3. *If a base process p is enabled, the advice ad of an aspect tpa that is woven into p is enabled.*

Proof. First, we prove that the activities in the entrance activity set A_e of ad are enabled. Then, we prove that the other activities in ad are enabled. The relations between the other activities and the activities in A_e are sequence, selection, concurrency, or iteration.

According to the analysis of the base-aspect weaving correctness in Section 3.2.2, the base-aspect weavings include six weaving operations. Here, we show the proof of flow weaving. According to the algorithm of condition-activity flow weaving in Algorithm 2, $\text{dom}(w.pc) \in p.C$, since p is enabled, there exists M' and $M'(\text{dom}(w.pc)) = 1$, then, by following the transition firing rule of Petri nets, A_e is enabled at marking M' ($M'[A_e >]$). The proofs of the other weaving operations are similar and are omitted for the sake of simplicity.

Next, we prove that the other activities in ad are enabled.

Sequence relation: Let $a \in ad$. A be an activity that has a sequence relation with A_e , and a can only be enabled after the execution of A_e , then $M[A_e >]$ and $\neg M[a >]$, but $M[A_e > M''[a >]$. At marking M' , A_e is enabled, and a is not enabled, but after A_e executes, the marking becomes M'' , and a is enabled.

Selection relation: Let $a_1, a_2 \in ad$. A be two activities that have selection relations with A_e , if $M[A_e > M''$, then $\neg M''[[a_1, a_2] >]$, but $M''[a_1 >]$ or $M''[a_2 >]$. After A_e executes, at marking M'' , a_1 and a_2 cannot execute at the same time because which one of them will execute is dependent on the execution result of A_e . But, a_1 and a_2 are enabled because tokens are resided at their input conditions. Q.E.D.

The proofs of the concurrency and iteration relations are similar and are omitted for the sake of simplicity.

5.1.3 | Completeness and correctness of the aspect-aspect weaving

Aspect-aspect interactions are necessary for establishing the desired overall behavior. However, if not treated properly, these interactions can give rise to interferences. Interferences are interactions that violate the specified constraints. The generic constraints of aspect-aspect weaving that Kniesel and Bardey defined are completeness and correctness.^{38,39} Completeness states that in the aspect-aspect weaving result, every aspect effect must be applied at all pointcuts. Correctness states that in the weaving result, every aspect effect must be applied only at the pointcuts. The following theorems state the completeness and correctness of aspect-aspect weaving.

Theorem 4. (Completeness of aspect-aspect weaving). Let $TPA = \{tpa_1, \dots, tpa_j\}$ be a set of aspects. If any two aspects in TPA have a joint pointcut and are woven together sequentially based on their data dependence relation, the aspect-aspect weaving has completeness.

Proof. According to Theorem 3, when the aspects in TPA are woven into the base processes, they are all enabled. Let tpa_1 and tpa_2 be two aspects in TPA that have data dependence relations. If they are woven together sequentially based on their data dependence relations, there exists M' , $M'[ad_1. A >]$ and $\neg M'[ad_2. A >]$, but $M'[ad_1. A > M''[ad_2. A >]$. $ad_2. A$ is enabled after $ad_1. A$ executes and obtains the resources from tpa_1 . Thus, $ad_2. A$ is not only enabled but also has the resources that it needs to let its effect be applied at the pointcut. Q.E.D.

Theorem 5. (Correctness of aspect-aspect weaving). Let $TPA = \{tpa_1, \dots, tpa_j\}$ be a set of aspects. If any two aspects in TPA have a joint pointcut and are woven together in a selection relation based on their control dependence relation, the aspect-aspect weaving has correctness.

Proof. According to Theorem 3, when the aspects in TPA are woven into the base processes, they are all enabled. Let tpa_1, tpa_2 , and tpa_3 be three aspects in TPA that have a control dependence relation. tpa_2 and tpa_3 are in a selection relation, that is, only one of them can execute. Whether tpa_2 executes or tpa_3 executes is dependent on the execution result of tpa_1 . If $M'[ad_1. A > M''$, then $\neg M''[[ad_2. A, ad_3. A] >]$, but $M''[ad_2. A >]$ or $M''[ad_3. A >]$. Only one of $ad_2. A$ and $ad_3. A$ is enabled after $ad_1. A$ executes and obtains the resources. Thus, at the pointcut, the aspect effect of $ad_2. A$ and $ad_3. A$ can only be applied in accordance with the execution result of tpa_1 . Q.E.D.

5.2 | Comparisons of modeling effectiveness

The modeling effectiveness is compared in the following. Before comparison, the project team conducted a thorough check of the definitions of the processes and assessed the performance of the aspect-oriented modeling. The checked processes included the base processes, the advices in aspects, and the trustworthiness processes. The checking procedure was performed automatically first and was followed by discussion within the project team. When the base processes and aspects were defined, the model checking was used to check the structural and dynamic properties automatically. During the weaving of the aspects, the weaving correctness was checked according to the aspect-aspect correctness and base-aspect correctness. When the woven trustworthiness processes were generated, their definitions were discussed within the project team to better assure the process trustworthiness. The performance of the aspect-oriented modeling was also assessed.

5.2.1 | Performance assessment of aspect-oriented modeling

In our performance assessments, we have considered the following:

- (1) The number of trustworthiness goals and soft goals. Intuitively, the greater the number of trustworthiness goals and soft goals, the more complex the reasoning, and hence, the more time that is required for the reasoning operations.
- (2) The size of the process model in terms of the number of activities. Intuitively, the greater the number of aspects, the more the codes increase, and the greater the number of activities, thus increasing the time needed for the correctness analysis.
- (3) The dependence relations of the weaving aspects. Intuitively, the weaving of the aspects is expected to introduce additional complexity due to the additional dependencies they add to the base process.

The first and second factors were taken into account by analyzing the performance using the case study models and randomly generated models of different sizes. When randomly generating 10 models, we avoided the risk of creating incorrect models by introducing a validity check of the model before executing the analysis operation. To minimize the impact of external factors on our results, each analysis operation was executed 10 times for each experiment to average values. The third factor was considered by analyzing the performance of the different dependencies. Three categories of dependencies were evaluated: aspect-aspect data dependence, aspect-aspect control dependence, and base-aspect dependence, which correspond to the weaving mechanism discussed above.

The evaluation results for the first factor are the runtimes of the TRM reasoning. Since zChaff has had success in solving problems with more than one million variables and 10 million clauses,³³ the total run times were all non-measurable (the total run times of our case study model are shown in Figure 12A,B) for all of our models, including the case study model and randomly generated models. The variables and clauses of our case study model were 58 and 65. The TG_activities we collected and listed in Table 2 were 120. Therefore, we generated models with 60 to 600 clauses and used our modified zChaff to reason. The total run times were always non-measurable.

The evaluation results for the second and the third factors are similar. The size of the process model in terms of adding aspects increased in the model codes. These increased codes were aspect definitions, aspect weaving configuration data, and XML tags. However, the correctness analysis time did not increase, and the total run time was approximately 0.1 to 0.4 seconds. Even when generating models with different dependence relations, the analysis time remained at this scale.

5.2.2 | Trustworthiness improvement of TROSP

TG_activities are designed to improve the trustworthiness of software by integrating them into the software processes. In the case study of the SIS software, 44 TG_activities were designed and integrated into the base processes at different levels. In practice, are TG_activities effective in trustworthiness improvement? To compare the changes in the trustworthiness of the SIS software, we collected project data for 2 months to assess the effects of the TG_activities. As an example, we use the activity *Code Review* as one of our evaluation objects. During the evolution development of the SIS software, after the execution of *Evolution Test* (see Figure 13), all developed codes were sent to the *Code Review* activity. *Code Review* is a TG_activity in which a set of program code is examined systematically by one or more peer reviewers. The intent is to find mistakes overlooked in software development and to improve the overall quality of the software. To compare the trustworthiness changes, we use *defect rate* to represent the trustworthiness. A lower *defect rate* indicates higher trustworthiness.

During the evolution procedure of the SIS software, the first 17 weeks were the phases of the evolution requirements analysis and evolution design. From the 18th week, evolution development began, and the evolution codes were submitted to the *Code Review* activity. From the 18th week to the 25th week, the defect rates before and after the execution of the *Code Review* activity were collected and are listed in Figure 15A. Figure 15B depicts the improvement result of these defect rates. Using these 2 months of data, we simulate the defect rate changes in the future 115 weeks to see the trend of defect rate changes. Figure 15C shows the simulation result. The upper line is the defect rate before *Code Review*, and the lower line is the result after *Code Review* is executed.

As another example, *Requirements V&V* is another TG_activity that was woven after *Proposal for Changes* (see Figure 14). This TG_activity analyzes the evolution requests from *Proposal for Changes* to decide which are valid requests and to make a priority list for these requests. The elimination of invalid and duplicate evolution requests reduces the invalid efforts of the project team. The priority list helps the project team reduce rework efforts and improve software quality. Figure 16A compares the number of valid requests before and after *Requirements V&V*. From the 18th week to the 21st week, invalid and duplicate requests were analyzed by the project team to verify the effectiveness of *Requirements V&V*. From the 22nd week to the 25th week, because the *Requirements V&V* was effective, the invalid and duplicate requests were no longer analyzed. We compared the defect rates in these 2 months and found that the number of invalid and duplicate requests increased with the defect rate. We used the same simulation method to compare the defect rate changes in the future 115 weeks. Figure 16B shows the simulation result. The upper line is the defect rate when *Requirements V&V* is not executed, whereas the lower line is the result when *Requirements V&V* is executed. The results show a decrease on the defect rate after *Requirements V&V*.

5.3 | Limitation

Although we got a positive result from the case study, we still found two limitations that are needed to be solved.

First, a knowledge base for TRM modeling has been created to provide NFRs, TG_activities, and their decomposition, implementation, and contribution relations. However, in the case study, only some of the TG_activities were chosen in the final TRM because too many additional activities may extend the duration of the project and cause complexity. Therefore, the project team discussed tailoring the TRM in the first four meetings and continued these discussions for many times during the evolution development. Based on the results of this case study, a new architecture for the knowledge base is proposed. In the new architecture, a general knowledge base, a domain knowledge base, and an individual knowledge base will be designed for better support of the modeling and tailoring of TRMs.

Second, the TPAT tool is useful in helping users weave the aspects and analyze the weaving correctness but not in defining the aspects. When defining an aspect, the pointcuts and weaving positions of the aspect rely on the base processes. A better understanding of the base processes is important. Therefore, in the case study, all activities in the base processes, as well as the input and output artifacts, were analyzed before defining

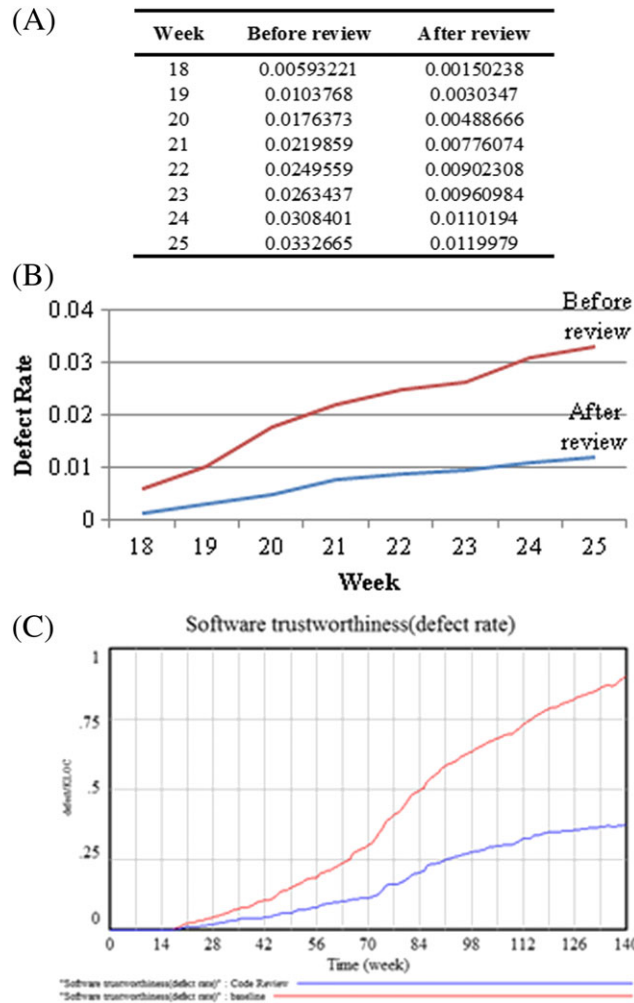


FIGURE 15 Trustworthiness improvement of code review activity: A, comparison of the defect rate; B, improvement results for the defect rate; C, simulation of the code review activity

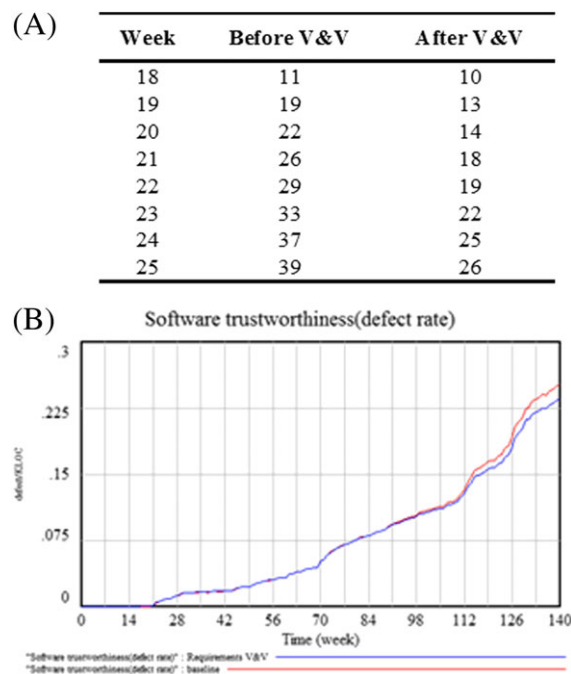


FIGURE 16 Trustworthiness improvement of the Design V&V activity: A, comparison of the number of valid; B, simulation of Requirements V&V activity

the aspects. In addition, an agile process was used to deliver the evolution software in every sprint, and the weaving of the aspects could be adjusted in the next sprint. However, providing better support for defining the aspects is a problem that remains to be solved.

6 | RELATED WORK

6.1 | Trustworthiness requirements modeling and reasoning

Goal-oriented requirement engineering is probably the most popular method for modeling and analyzing NFRs. The NFR framework,⁴⁰ knowledge acquisition in automated specification,⁴¹ i* families (including i* model, Tropos, Goal-oriented Requirement Language),⁴²⁻⁴⁴ and Techne⁴⁵ are the basic methods and languages of goal-oriented requirement engineering. Subsequently, more goal-oriented approaches and frameworks have been formulated. The focus of this research has been mostly on modeling and reasoning, with the aim of assisting software engineers in analyzing NFR relations. Among these studies, only Zhu et al⁴⁶ used soft goal interdependency graphs to address NFR correlation analysis in trustworthy software. To analyze the positive and negative contribution relations of the NFRs and operation tasks, either forward or backward reasoning is used. In backward reasoning, Giorgini and co-authors first used a SAT solver for goal models to assign strategies that satisfy the desired status of the NFRs.⁴⁷⁻⁴⁹ Subsequently, Sebastiani et al,³² Horkoff and colleagues,^{31,50} and Ernst et al⁵¹ improved this backward reasoning technique. Their approaches are effective for NFRs analysis. By adapting their work, we defined our propagation axioms to find the trustworthiness goal-oriented activities.

6.2 | Trustworthy software and software processes

As noted previously, software development and evolution are process-intensive undertakings.¹ To improve the trustworthiness of software, three types of process-oriented methods have been proposed. They are software process improvement, phase-specific software development, and process quality assurance.

Software process improvement is intended to define a method for analyzing, quantifying, and enhancing the efficiency and quality of the software process involved in software production and delivery. The main objective is to make software processes repeatable and consequently minimize the number and magnitude of errors in the processes. Model-driven process improvement and measure-driven process improvement are two different software process improvements. The best known model-driven process improvement models are capability maturity model (CMM), and its variants, CMM integration (CMMI) and secure systems engineering-CMM.⁵² Define measure analyze improve control (DMAIC) and define measure analyze design verify (DMADV) of Six Sigma are representatives of measure-driven software process improvement.

Phase-specific software development methodologies tend to be more detailed in that they describe not only what activities should occur during a particular life cycle phase but also how those activities should be performed. Microsoft's SDLC,¹⁰ software assurance maturity model of open web application security project,²⁹ Secure Software Inc.'s comprehensive lightweight application security process,⁵³ McGraw's Touchpoints,⁵⁴ and correctness by construction^{55,56} define a wide variety of activities an organization could engage in to reduce risks and increase software assurance. Our TROSP modeling is also a phase-specific software development method. A number of TR-oriented activities are included in the software development life cycle. Furthermore, formal correctness is defined, examined, and proved for process quality assurance.

Software quality cannot be ensured without guaranteeing the quality of the process by which the software is developed. Trusted Software methodology,⁶ trustworthy process management framework,¹ and transformable process modeling²⁰ specify the process attributes that contribute to enhanced software trustworthiness.

6.3 | Aspect-oriented Petri nets and correctness

To provide a detailed TROSP modeling approach and to control the correctness of the modeling, we proposed to use aspect-oriented Petri nets to model and provide correctness.

Some previous studies had proposed the approach of modeling aspect-oriented Petri nets. Roubtsova and Aksit proposed the Aspect Petri Net notation.⁵⁷ Xu and Nygard proposed aspect-oriented Predicate-Transition Petri net.⁵⁸ Molderez et al⁵⁹ presented the aspect-oriented extension to Petri nets. Only Guan et al⁶⁰ used Xu and Nygard's aspect-oriented Predicate-Transition Petri net⁵⁸ and Nagy's aspect relations definitions⁶¹ to resolve the problems with aspect-aspect correctness.

In aspect-oriented methods, aspect-aspect correctness has also been studied by many researchers for many years, such as Constantinides et al's moderator pattern,⁶² Kiczales et al's dominates modifier in AspectJ,⁶³ Douence et al's general aspect independences,^{64,65} Pawlak et al's CompAr,⁶⁶ Nagy et al's ordering and control constraints,⁶¹ Durr et al's semantics definition of advices on an abstract resource model,⁶⁷ Kniessel and Bardey's incorrect and incomplete weaving,^{38,39} and Dinkelaker et al's feature interactions.⁶⁸

In summary, in these studies, aspect-aspect correctness is determined by the correct interdependent aspects. This type of correctness is ensured by first analyzing the interdependencies among the aspects and then weaving them based on the interdependencies. We used the same method in this paper. However, there has been no study of base-aspect correctness. As mentioned above, the analysis and control of base-aspect correctness should also be considered.

7 | CONCLUSIONS

A trustworthy software process is “a process that is capable of producing a range of trustworthy software products”.²⁰ In this paper, our aims were to explore theories and methods for enhancing, improving, and innovating software process techniques and to support the development and production of large-scale complex trustworthy software. We investigated the mechanism for integrating trustworthiness goal-oriented activities into software processes and developed effective tools to aid the modeling of TROSPs. The major contributions of this paper are the following:

- We proposed an approach to modeling TROSPs. These processes deliver trustworthiness by introducing trustworthiness goal-oriented activities (TG_activities) and ensuring the correctness of the process modeling.
 - Since the enforcement of a TG_activity may undermine the satisfaction of the other TRs, goal-oriented modeling and reasoning for TRs were provided to find TG_activities that satisfy multiple TRs.
 - When integrating TG_activities into software processes, aspect-oriented modeling techniques were adopted. Correctness of the integration between multiple TG_activities and between TG_activities and software processes was analyzed, and correct integration methods were designed. Errors or mistakes of process modeling can be prevented.
- Based on the aspect-oriented modeling techniques, plug-ins for generating TROSPs provide flexibility and maintainability in an organized manner. The proposed approach of process modeling can adequately match the evolutions and changes in the TRs.

The evaluations in Section 5 proved the correctness of our modeling and reasoning and demonstrated the modeling efficiency and feasibility of improving trustworthiness. Limitations were also discussed. Corresponding future work will provide better support for TR modeling and for defining aspects. In addition, guidance will be provided to industry on practical aspects of the modeling to allow further validation with a greater variety of realistic cases.

ORCID

Xuan Zhang  <http://orcid.org/0000-0003-2929-2126>

REFERENCES

1. Yang Y, Wang Q, Li MS. Process trustworthiness as a capability indicator for measuring and improving software trustworthiness. International Conference on Software Process (ICSP'09), Vancouver, Canada. Springer: Berlin. 2009;5543(5):389-401.
2. McLean J. Trustworthy software why we need it, why we don't have it, how we can get it. International Computer Software and Applications Conference (COMPSAC'06), Chicago. IEEE Computer Society: Silver Spring MD. 2006;1(9):32-33.
3. Van Der Aalst WMP. The application of petri nets to workflow management. *Journal of Circuits, Systems and Computers*. 1998;8(1):21-45.
4. Reisig W. *Petri Nets: An Introduction*. Berlin: Springer Verlag; 1985.
5. Li CY, Ge JD, Huang LG, et al. Software cybernetics in BPM: modeling software behavior as feedback for evolution by a novel discovery method based on augmented event logs. *The Journal of Systems and Software*. 2017;124:260-273.
6. Amoroso E, Taylor C, Watson J, Weiss J. A process-oriented methodology for assessing and improving software trustworthiness. In the Proceedings of the 2nd ACM Conference on Computer and Communications Security (CCS'94), 1994:39-50.
7. United States Department of Defense. Trusted Computer System Evaluation Criteria (TCSEC). DoD 5200.28-STD. Washington: Department of Defense. <http://www.cerberussystems.com/INFOSEC/stds/d520028.htm>, 1985, 12.
8. IEC. International Electrotechnical Vocabulary—Part 192: Dependability (IEC 60050-192 Ed.1.0), 2015, 2.
9. Howard M, Leblanc D. *Writing Secure Code*. Microsoft Press; 2002.
10. Howard M, Lipner S. *The Secure Development Life-Cycle*. Microsoft Press; 2006.
11. Trusted Computing Group (TCG), 2007. TCG Specification Architecture Overview, Revision 1.4. <http://www.trustedcomputinggroup.org>.
12. Littlewood B, Strigine L. Software reliability and dependability: a roadmap. In: Finkelstein, A. (Ed.). *The Future of SE, ICSE'22, IEEE*, 2000:175-188.
13. Schmidt H. Trustworthy components—compositionality and prediction. *The Journal of Systems and Software*. 2003;65(3):215-225.
14. Neumann PG. *Principled Assuredly Trustworthy Composable Architectures*. Computer Science laboratory, SRI International: Project Report; 2004.
15. NSS2. Software 2015: a national software strategy to ensure U.S. security and competitiveness. <http://www.cnsoftware.org/nss2report/>, 2005, 4.
16. Bernstein L, Yuhas C. *Trustworthy Systems through Quantitative Software Engineering*. 1 New York, Silver Spring MD: Wiley-IEEE Computer Society Press; 2005.
17. Hasselbring W RR. Toward trustworthy software systems. *Computer*. 2006;39(4):91-92.
18. Miller A, Mclean J, Saydjari O, Voas J. Compsac panel session on trustworthy computing. COMPSAC'06: Proceedings of 30th Annual International Computer Software and Applications Conference, Chicago IL, vol. 1. IEEE Computer Society: Silver Spring MD, September 2006;31.
19. Trustie. Software Trustworthiness Classification Specification (TRUSTIE-STC v 1.0), 2009, <http://www.trustie.net/>.
20. Zhang H, Kitchenham B, Jeffery R. Toward trustworthy software process models: an exploratory study on transformable process modeling. *Journal of Software: Evolution and Process*. 2012;24(7):741-763.
21. Dalkey N, Helmer O. An experimental application of the Delphi method to the use of experts. *Management Science*. 1963;9(3):458-467.
22. Lyu R. *Handbook of Software Reliability Engineering*. IEEE Computer Society: Washington; 1996.

23. Musa D. *Software Reliability Engineering [M]*. Columbus: McGraw-Hill; 1999.
24. Anderson R. *Security Engineering*. Hoboken, NJ: John Wiley & Sons; 2008.
25. Ericson A. *Hazard Analysis Techniques for System Safety*. Hoboken, NJ: John Wiley & Sons; 2005.
26. United States Department of Defense. Standard practice for system safety, MIL-STD-882D. <http://www.system-safety.org/Documents/MIL-STD-882D.pdf>, 2000, 2.
27. Nielsen J. *Usability Engineering*. New York: Elsevier Ltd Oxford; 1994.
28. Boehm B, In H. Identifying quality-requirement conflicts. *IEEE Software*. 1996;13(2):25-35.
29. OWASP (the open web application security project). Software assurance maturity model—a guide to building security into software development. <http://www.opensamm.org/>. 2009.
30. Li T. *An Approach to Modelling Software Evolution Processes*. Berlin: Springer-Verlag; 2008.
31. Horkoff J, Yu E. Finding solutions in goal models: an interactive backward reasoning approach. In: *Conceptual Modeling—ER 2010*. Berlin Heidelberg: Springer; 2010:59-75.
32. Sebastiani R, Giorgini P, Mylopoulos J. *Simple and Minimum-Cost Satisfiability for Goal Models*. Berlin Heidelberg: Advanced Information Systems Engineering, Springer; 2004:20-35.
33. Princeton University, zChaff 2007.3.12. <http://www.princeton.edu/~chaff/zchaff.html>.
34. Choe Y. prop2chf.py. CSCE 625: introduction to machine learning. <http://faculty.cs.tamu.edu/ioerger/cs625-fall11/prop2cnf.py>.
35. Blair GS, Blair L, Rashid A, Moreira A, Araújo J, Chitchyan R. *Engineering Aspect-Oriented Systems*. Addison-Wesley, Boston: Aspect-Oriented Software Development; 2005:379-406.
36. Schauerhuber A, Schwinger W, Kapsammer E, Retschitzegger W, Wimmer M, Kappel G. *A Survey on Aspect-Oriented Modeling Approaches*. Relatorio tecnico: Vienna University of Technology; 2007.
37. Imperial College London. Platform independent petri net editor (PIPE) v4. <https://sourceforge.net/projects/pipe2/?source=navbar>, 2013.
38. Kniessel G, Bardey U. An analysis of the correctness and completeness of aspect weaving. The 13th Working Conference on Reverse Engineering (WCRE'06), IEEE, 2006:324-333.
39. Kniessel G. *Detection and Resolution of Weaving Interaction*. Transactions on Aspect-Oriented Software Development, Springer, Berlin Heidelberg; 2009:135-186.
40. Mylopoulos J, Chung L, Nixon B. Representing and using nonfunctional requirements: a process-oriented approach. *IEEE Trans on Software Engineering*. 1992;18(6):483-497.
41. Van Lamsweerde A, Darimont R, Letier E. Managing conflicts in goal-driven requirements engineering. *IEEE Trans on Software Engineering*. 1998;24(1):908-926.
42. Yu E. Towards modeling and reasoning support for early-phase requirements engineering. The 3rd IEEE International Symposium on Requirements Engineering, 1997: 226-235.
43. Castro J, Kolp M, Mylopoulos J. Towards requirements-driven information software engineering: the Tropos project. *Information Software*. 2002;27(6):365-389.
44. Amyot D, Mussbacher G. URN: towards a new standard for the visual description of requirements. In: *The Telecommunications and Beyond: The Broader Applicability of SDL and MSC*. Berlin, Heidelberg: Springer-Verlag; 2003:21-37.
45. Jureta IJ, Borgida A, Ernst NA, Mylopoulos J. Techne: towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. The 18th IEEE international requirements engineering conference, Sydney. 2010;11:115-124.
46. Zhu MX, Luo XX, Chen XH, Wu DD. A non-functional requirements tradeoff model in trustworthy software. *Information Science*. 2012;191:61-75.
47. Giorgini P, Mylopoulos J, Nicchiarelli E. *Reasoning with goal models*. *Conceptual modeling—ER 2002*. Berlin Heidelberg: Springer; 2002:167-181.
48. Giorgini P, Mylopoulos J, Nicchiarelli E, Sebastiani R. Formal reasoning techniques for goal models. *Journal on Data Semantics*. 2003;11(1):1-20.
49. Giorgini P, Mylopoulos J, Sebastiani R. Goal-oriented requirements analysis and reasoning in the Tropos methodology. *Eng Appl Artif Intel*. 2005;18(2):159-171.
50. Horkoff J, Yu E. Interactive goal model analysis for early requirements engineering. *Requirements Engineering Springer*. 2016;21(1):29-61.
51. Ernst NA, Mylopoulos J, Borgida A, Jureta IJ. Reasoning with optional and preferred requirements. In: *Conceptual Modeling—ER 2010*. Berlin Heidelberg: Springer; 2010:118-131.
52. CMU. Systems Security Engineering Capability Maturity Model SSE-CMM: Model Description Document, Version 3.0, 2003.
53. Secure Software Inc. The CLASP application security process. http://www.ida.liu.se/~TDDC90/papers/clasp_external.pdf, 2005.
54. McGraw G. *Software Security: Building Security in*. Addison-Wesley Professional; 2006.
55. Hall A, Chapman R. Correctness by construction: developing a commercial secure system. *IEEE Software*. 2002;1:18-25.
56. Hall A. Correctness by construction: integrating formality into a commercial development process. In: *Formal Methods—Getting IT Right (FRM 2002)*, LNCS 2391. Springer Verlag; 2002:224-233.
57. Roubtsova EE, Aksit M. Extension of petri nets by aspects to apply the model driven architecture approach. The 1st International Workshop on Aspect-Based and Model-Based Separation of Concerns in Software Systems (ABMB). Nuremberg, Germany, 2005.
58. Xu DX, Nygard K. Threat-driven modeling and verification of secure software using aspect-oriented petri nets. *IEEE Transactions on Software Engineering*. 2006;32(4):265-278.
59. Molderez T, Meyers B, Janssens D, Vangheluwe H. Towards an aspect-oriented language module: aspects for petri nets. The 7th workshop on Domain-Specific Aspect Languages. *ACM*. 2012;3:21-26.
60. Guan LW, Li X, Hu H, Lu J. A petri net-based approach for supporting aspect-oriented modeling. *Frontiers of Computer Science in China*. 2008;2(4):413-423.

61. Nagy I, Bergmans L, Akşit M. Composing aspects at shared join points. International Conference NetObjectDays (NODE2005), Lecture notes in Computer Science, Springer, 2005:69-84.
62. Constantinides CA, Bader A, Elrad T. An aspect-oriented design framework for concurrent systems. In: *The ECOOP'99 Workshop on Aspect-Oriented Programming*. Portugal: Lisbon; 1999:302-311.
63. Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold WG. An overview of aspectJ. European Conference on Object-Oriented Programming (ECOOP), 2001:327-353.
64. Douence R, Fradet P, Südholt M. *A Framework for the Detection and Resolution of Aspect Interactions*. Berlin Heidelberg: Generative Programming and Component Engineering, Springer; 2002:173-188.
65. Douence R, Fradet P, Südholt M. Composition, reuse and interaction analysis of stateful aspects. The 3rd international conference on aspect-oriented software development, ACM, 2004:141-150.
66. Pawlak R, Duchien L, Seinturier L. *CompAr: Ensuring Safe Around Advice Composition*. Berlin Heidelberg: Formal Methods for Open Object-Based Distributed Systems, Springer; 2005:163-178.
67. Durr P, Stajien T, Bergmans L, Akşit M. Reasoning about semantic conflicts between aspects. The 2nd European Interactive Workshop on Aspects in Software (EIWAS'05), 2005:10-18.
68. Dinkelaker T, Erradi M, Ayache M. Using aspect-oriented state machines for detecting and resolving feature interactions. *Computer Science and Information Systems*. 2012;9(3):1045-1074.

How to cite this article: Zhang X, Wang X, Kang YN. Trustworthiness requirement-oriented software process modeling. *J Softw Evol Proc*. 2018;30:e1991. <https://doi.org/10.1002/smr.1991>