A New Trust Chain Security Evaluation Model and Tool

Wei Hu, Dongyao Ji(✉), Ting Wang, and Gang Yao

State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
jidongyao@is.ac.cn

Abstract. We've build a model of trust chain, and developed TCSE, a tool for estimating the security properties of the trust chain. The highlight of TCSE is that it can generate a probabilistic finite state automaton and verify or calculate four security properties of a trust chain following our algorithms. These properties are: credibility, usability, restorability and conformity. With these four values of a trust chain, we can estimate the security of a trusted computer (a computer with a trusted computing module). Using this tool, an ordinary user with the help of the Common Vulnerability Scoring System (CVSS) from which one can easily get the needed parameters can figure out these four properties quickly. This tool can be used in the area where the security of trusted computers are needed to be precisely quantized.

Keywords: Trusted computing · Trust chain · Model checking · Probabilistic finite state automaton · Probabilistic computation tree logic

1 Introduction

Trusted computing is a trend of information security technology and it has been used on a large scale at present. One key aspect of trusted platform is its ability to record the trust relationship among components that make-up the trusted platform. Trust is the expectation that a device will behave in a particular manner for a specific purpose. When one trusted component measures the trustworthiness of a second component, trust is transferred transitively from the first component to the second. That's the principle of the trust chain and an example is showed in Fig. 1. The implementation of the trust chain is up to its vendors, so it's possible that the implementation of it doesn't conform to its specifications and may lead to some security problems. The key technology of trust chain testing is the conformance testing. Xu [1] focused on the behavior characters of specifications of trust chain, and proposed a conformance testing framework for it based on labeled transition system. Fu [2] built a formal model of trust chain specifications based on finite state machines and analyzed the test sequence generation procedure with unique input/output sequence. Zhan [3] gave a conformance testing model of TPM based on state machine model. But there is no model that based

J. Tian et al. (Eds.): SecureComm 2014, Part II, LNICST 153, pp. 382–391, 2015.
DOI: 10.1007/978-3-319-23802-9_30

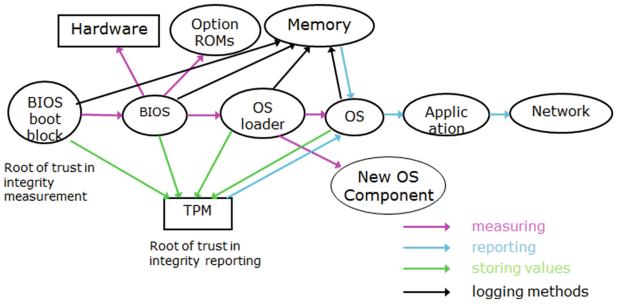

Fig. 1. Example of trust chain emanating from the trust root

on probabilistic finite state automaton to describe trust chain, and there is no tool that can calculate the values of the security properties of the trust chain.

We build a model of a trust chain with PFSA (probabilistic finite state automaton) and give four security properties in the form of PCTL (probabilistic computation tree logic), and present TCSE, a tool for estimating the trust chains on trusted computers. We analyse four characteristics of the trust chain: credibility, usability, restorability, conformity and invoke PRISM [4] (Probabilistic Symbolic Model Checker) to calculate the values of them. The reason why we choose these four properties to be verified and calculated is that they play the most important roles in the security of the trust chain. This paper describes the complete tool features and the implementation details of TCSE developed by us.

2 Formal Model of Trust Chain

2.1 Probabilistic Finite State Automaton

Markov chains [5] is a kind of probabilistic finite state automaton, it behaves as transition systems with the only difference that nondeterministic choices among successor states are replaced by probabilistic ones. That is to say, the successor state of state s, say, is chosen according to a probability distribution. This probability distribution only depends on the current state s, and not on, e.g., the path fragment that led to state s from some initial state. Accordingly, the system evolution does not depend on the history (i.e., the path fragment that has been executed so far), but only on the current state s. This is known as the memoryless property. A (discrete-time) Markov chain is a tuple M = (S, P, l_{init}, AP, L) where:

- S is a countable, nonempty set of states,
- P: S × S → [0, 1] is the transition probability function such that for all states s′: $\sum_{s' \subset S} P(s, s') = 1$,
- l_{init}: S → [0,1] is the initial distribution, such that $\sum_{s \subset S} l_{init}(s) = 1$, and

– AP is a set of atomic propositions and
– L: S $\rightarrow 2^{AP}$ a labeling function.

Markov chains' ability of expression is very strong, so we use Markov chains to model trust chain, Fig. 2 is a markov model we built for the trust chain. Considering many aspects that can affect the properties we considered, we use variables to describe the probabilities.

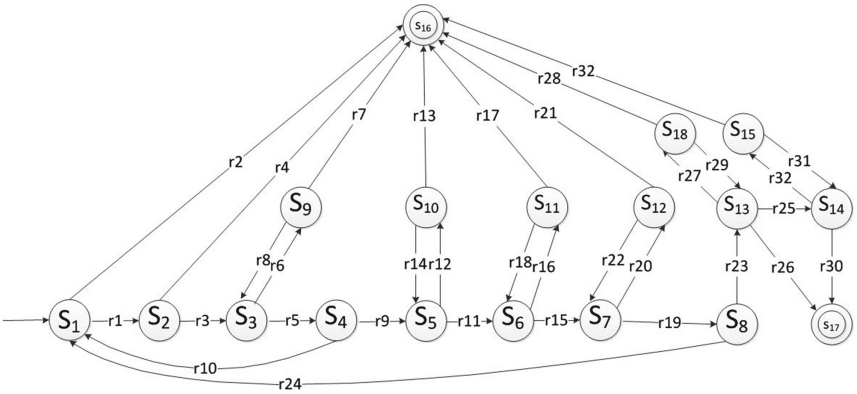

Fig. 2. The PFSM of a trust chain

In Fig. 2 showed above, there are 17 states in it. These states are described below. S_1:system boot, S_2:CRTM power-on self-test, S_3:BIOS measurement, S_4:BIOS user authentication, S_5:key component and configuration measured by BIOS, S_6:measurement of MBR and OS Loader, S_7:measurement of OS kernel and drivers and system files, S_8:OS user authentication, S_9:BIOS recovery, S_{10}:core component and configuration recovery, S_{11}:recovery of MBR and OSLoader, S_{12}:recovery of OS kernel and drivers, S_{13}:static integrity measurement of applications, S_{14}:dynamic integrity measurement of applications, S_{15}:dynamic recovery of applications, S_{16}:untrusted state, S_{17}:trusted state, S_{18}:static recovery of applications. We can get the state transition probabilities of the model from CVSS, since the semantics of these transitions are very clear. CVSS is is a vulnerability scoring system designed to provide an open and standardized method for rating IT vulnerabilities. With the help of it, we can achieve the values from r1 to r32. With these parameters, we can figure out some security property values.

2.2 Trust Chain Probabilistic Model Checking

CTL is an important branching temporal logic that is sufficiently expressive for the formulation of an important set of system properties. It was originally used by Clarke and Emerson and (in a slightly different form) by Queille and Sifakis

[6] for model checking. More importantly, it is a logic for which efficient and as we will see rather simple model-checking algorithms do exist. It has a two-stage syntax where formulae in CTL are classified into state and path formulae. The former are assertions about the atomic propositions in the states and their branching structure, while path formulae express temporal properties of paths. Compared to LTL formulae, path formulae in CTL are simpler: as in LTL they are built by the next-step and until operators, but they must not be combined with Boolean connectives and no nesting of temporal modalities is allowed.

Probabilistic computation tree logic (PCTL, for short) is a branching-time temporal logic, based on the logic CTL. A PCTL formula formulates conditions on a state of a Markov chain. The interpretation is Boolean, i.e., a state either satisfies or violates a PCTL formula. The logic PCTL is defined like CTL with one major difference. Instead of universal and existential path quantification, PCTL incorporates, besides the standard propositional logic operators, the probabilistic operator $P_J(\varphi)$ where φ is a path formula and J is an interval of [0, 1]. The path formula φ imposes a condition on the set of paths, whereas J indicates a lower bound and/or upper bound on the probability. The intuitive meaning of the formula $P_J(\varphi)$ in state s is: the probability for the set of paths satisfying φ and starting in s meets the bounds given by J. The probabilistic operator can be considered as the quantitative counterpart to the CTL path quantifiers $\exists$ and $\forall$. The CTL formulae $\exists\varphi$ and $\forall\varphi$ assert the existence of certain paths and the absence of paths where a certain condition does not hold respectively. They, however, do not impose any constraints on the likelihood of the paths that satisfy the condition ϕ. Later on in this section, the relationship between the operator $P_J(\varphi)$ and universal and existential path quantification is elaborated in detail.

PCTL state formulae over the setAP of atomic propositions are formed according to the following grammar:

$$\phi ::= true|a|\neg\phi|\phi_1 \wedge \phi_2|P_{\lhd p}(\varphi)$$

where a $\in$ AP, φ is a path formula and J $\subseteq$ [0, 1] is an interval with rational bounds. PCTL path formulae are formed according to the following grammar:

$$\varphi ::= \bigcirc\phi|\phi_1 \bigcup \phi_2$$

Let a $\in$ AP be an atomic proposition, M=(S, P, l_{init}, AP, L) be a Markov chain, state s $\in$ S, Φ, Ψ be PCTL state formulae, and φ be a PCTL path formula. The satisfaction relation $\models$ is defined for state formulae by

$$\begin{array}{ll} s \models a & \text{iff } a \in L(s) \\ s \models \neg\phi & \text{iff } not(s \models \phi) \\ s \models \phi \wedge \psi & \text{iff } s \models \phi \text{ and } s \models \psi \\ s \models P_{\lhd p}(\varphi) & \text{iff } Prob(s \models \varphi) \lhd p \end{array}$$

With PCTL we can describe the four security properties. The formulas used below are based on: $Prob(s \vDash \varphi) = Pr\{\pi \in Paths(s)|\pi \vDash \varphi\}$, Paths(s) stands for paths which use state s as the initial state. We provide four formulas and use PRISM to calculate these security property values.

2.3 The Computational Formulas of the Four Security Properties

Credibility. In the calculation of credibility, we need to use cylinder set. Let $\pi' = s_0 \cdot s_1 \cdot \cdots \cdot s_n \in Paths_{fin}(M)$ the cylinder set of π' is defined as:

$$Cyl(\pi') = \{\pi \in Paths(M) | \pi' \in pref(\pi)\}$$

The cylinder set spanned by the finite path π' thus consists of all infinite paths that start with π'. the probabilities for the cylinder sets $Pr(Cyl(s_0 \cdot s_1 \cdots \cdot s_n)) = P(s_0 \cdot s_1 \cdot \cdots \cdot s_n)$, where $P(s_0 \cdot s_1 \cdot \cdots \cdot s_n) = \prod_{0 \le i < n} P(s_i, s_{i+1})$.

According to Fig. 2 state s_{17} is the final trusted state. So the credibility of the trust chain can be described as $Pr(\diamond s_{17})$. The following formula is used to calculate credibility. We use Pr($\diamond$ B) to stand for credibility of the trust chain and B stands for the final state that we expect the system to reach.

$$\begin{aligned} Pr(\diamond B) &= \textstyle\sum_{s_0 \cdots s_n \in Paths_{fin}(M) \cap (S \setminus B)^* B} Pr(Cyl(s_0 \cdots s_n)) \\ &= \textstyle\sum_{s_0 \cdots s_n \in Paths_{fin}(M) \cap (S \setminus B)^* B} P(s_0 \cdots s_n) \end{aligned}$$

In the formula above, we use $s_0 ... s_n$ to stand for paths in M (our trust chain model) and $s_0 ... s_{n-1} \notin B, s_n \in B$. So these paths can be expressed as:

$$Paths_{fin}(M) \cap (S \setminus B)^* B.$$

$Cyl(s_0 ... s_n)$ is the cylinder set [6] of finite path $s_0 ... s_n$. According to the definition of cylinder set, we have $Cyl(s_0 ... s_n) = \{\pi \in Paths(M) | s_0 ... s_{(n)} \in pref(\pi)\}$. According to this formula we can calculate the value of credibility.

Usability. Some kinds of attacks can make the trust chain lose its function. For example, in TOCTOU [8] (Time Of Check to Time Of Use) attack, an adversary can exploit the time difference between when software is measured and when it is actually used, to induce run-time vulnerabilities. We notice that the current TCG architecture only provides load-time guarantees. Integrity measurements are taken just before the software is loaded into memory, and it is assumed that the loaded in-memory software remains unchanged. However, this is not necessarily true. Another attack called the Cuckoo attack [9] happens when malware on the local machine may forward the user's messages to a remote TPM that the adversary physically controls. Thus, the user cannot safely trust the TPM's state, and hence can't trust the computer in front of him. Both of these attacks can make some measurements through the trust chain bypassed. Specific to these attacks, we build a new model that can take these situations into count. Then we can handle this probabilistic automation and figure out the usability.

Restorability. In the TCG specification, a trusted computer should provide a component to help it recover from bad states. For example, an OS Loader's backups can help it recover from damages. But in practice many manufacturers omit these components for the sake of cost reduction. So we figure out the final reliability differentials between the model in specification and the model of the user to gain the restorability value.

Conformity. In a complete trust chain, there are a lot of components to be measured, for example, CRTM, BIOS, OS Loader and so on. But some trusted computers may leave out some of these components and we don't know how much it influences our computers' security. So we use PPTL [10,11] (Propositional Projection Temporal Logic) to calculate this value. Let Prop be a countable set of atomic propositions. The formula P of PPTL is given by the following grammar:

$$P ::= p| \bigcirc P|\neg P|P_1 \bigvee P_2|(P_1, ..., P_m)prjP$$

where p ∈ Prop, P_1,..., P_m and P are all well-formed PPTL formulas, ◯(next) and prj (projection) are basic temporal operators. The abbreviations true, false, ∧, → and ↔ are defined as usual. In particular, true $\stackrel{def}{=}$ P∨¬P and false $\stackrel{def}{=}$ P∧¬P for any formula P. Also we have the following derived formulas:

$$\begin{array}{ll}
\varepsilon \stackrel{def}{=} \neg \bigcirc true & more \stackrel{def}{=} \neg\varepsilon \\
\bigcirc^0 P \stackrel{def}{=} P & \bigcirc^n P \stackrel{def}{=} \bigcirc(\bigcirc^{n-1} P) \\
len\ n \stackrel{def}{=} \bigcirc^n \varepsilon & skip \stackrel{def}{=} len\ 1 \\
\odot P \stackrel{def}{=} \varepsilon \vee \bigcirc P & P; Q \stackrel{def}{=} (P, Q)prj\ \varepsilon \\
\diamond P \stackrel{def}{=} true; P & \Box P \stackrel{def}{=} \neg \diamond \neg P \\
halt(P) \stackrel{def}{=} \Box(\varepsilon \leftrightarrow P) & fin(P) \stackrel{def}{=} \Box(\epsilon \leftrightarrow P) \\
keep(P) \stackrel{def}{=} \Box(\neg\varepsilon \rightarrow P) &
\end{array}$$

where ⊙(weak next), □(always), ◇(sometimes), and ;(chop) are derived temporal operators; ε(empty) denotes an interval with zero length, and $\bar{\varepsilon}$(more) means the current state is not the final one over an interval. Prj projection operation allows the characterization of different time granularity calculation process. To explain (P_1, ..., P_m) prj Q, requires two different time granularity state sequence: one is executed P_1, ..., P_m of the local sequence, and the other is the Q of the overall sequence of parallel execution. Visually speaking, Q and P_1, ..., P_m parallel execution on an interval and the interval Q state is only P_1, ..., P_m each interval the initial state and final state, as shown in Fig. 3, projection operation Allow Q, P_1, ..., P_m each independently, have the right to define its execution interval.

In order to figure out this value, we should first transform PPTL's properties into NF (Normal Form), and then make out the NFG (Normal Form Graph). If the NFG is an NFA (Nondeterministic Finite Automaton), we should transform

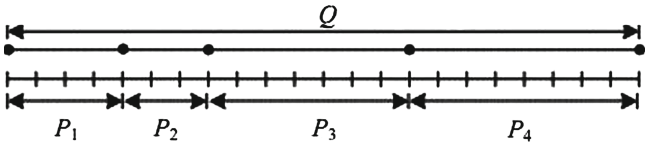

Fig. 3. Semantics of (P_1, P_2, P_3, P_4) prj Q

it into a DFA (Deterministic Finite Automaton). Then we can calculate the conformity with the inputs which should be the product of the model and its property which is the product of a Markov chain and a DFA..

The product of Markov chain and DFA is showed as follows:

Suppose M = (S, P, s_0, AP, L) is a Markov chain, A = (Q, 2^{AP}, δ,q_0, F) is a DFA, so the product of M and A M⊗A is a Markov chain: M⊗A = (S × Q, P′, s'_0, {accept}, L′) and in it:

$$L'(\langle s,q\rangle) = \begin{cases} \{accept\}, if\ q \in F \\ \emptyset, else \end{cases}$$

$$s'_0 = \begin{cases} \langle s_0,q\rangle, if\ q = \delta(q_0, L(s_0)) \\ 0, else \end{cases}$$

$$P'(\langle s,q\rangle,\langle s',q'\rangle) = \begin{cases} P\langle s,s'\rangle, if\ q' = \delta(q, L(s')) \\ 0, else \end{cases}$$

The path of M⊗A: $\pi =< s_0, q_1 >< s_1, q_2 >...$ is the combination between path of M: $s_0 s_1$.. and the path of A: $q_1 q_2$... In order to calculate the probabilistic of the path in the Markov chain M that satisfy the property of Q, which is $Pr^M(s_0 \vDash Q)$, the property Q of PPTL should be changed into NF(Normal Form), then the NFG(Normal Form Graph) can be drawn. Since the NFG is not a NFA, so it should be changed into a DFA. In order to calculate $Pr^M(s_0 \vDash Q)$, we can get the product of M and A, which is M⊗A. Then we can get the final formula as: $Pr^M(s_0 \vDash Q) = Pr^{M\otimes A}(s'_0 \diamond accept)$

3 Main Features and Implementation of TCSE

Figure 4 illustrate the architecture and the components of TCSE, the core features of it will be described detailedly below.

The Graphical User Interface (GUI). The GUI of TCSE provides users with functions to calculate their trusted computers' security properties. Users can refer to the CVSS [7] to ascertain the values of the parameters which are shown on the interface of TCSE. The GUI is designed with VC++6.0, and we

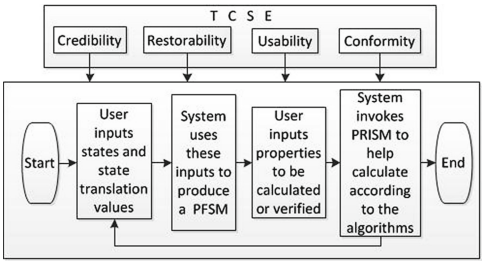

Fig. 4. The architecture of TCSE

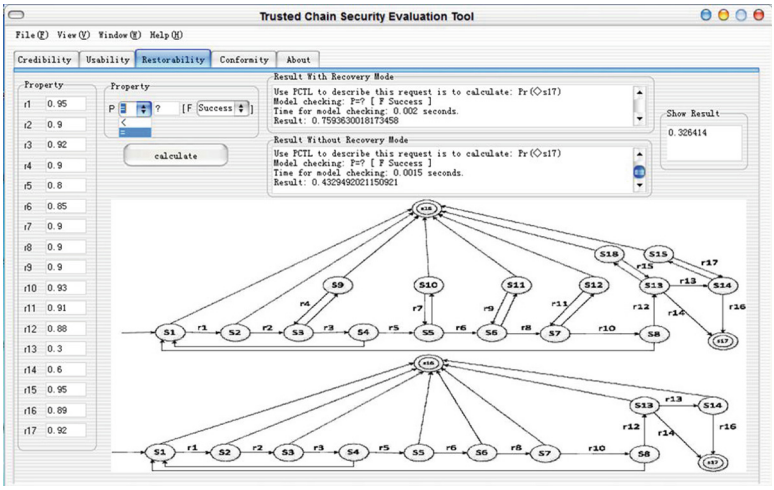

Fig. 5. The property of credibility

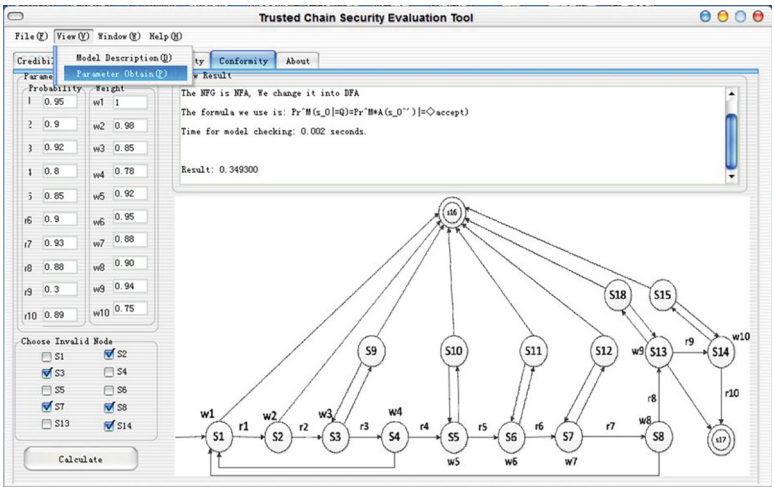

Fig. 6. The property of conformity

use a skin library to make the interface more artistic. We give a standard model of a trust chain provided in the TCG (Trusted Computing Group) specification with all sides considered. So users can compare their trust chains with this model and then get the parameters from the CVSS, these parameters are used to make up the PFSM (Probability Finite State Machines). After these parameters are given, users can verify or calculate any expressions in the form of PCTL.

Implementation of TCSE. We use four classes derived from class CPropertyPage in MFC (Microsoft Foundation Class) to deal with each property. Each

property page (is also a kind of dialog box in MFC) provides interface with users and users can get the results and some detailed calculation information from the interface. The system can also produce state translation graphs and some explanations to describe them, so that users can understand the models established by us following the specifications of TCG. By supplying the parameters (including the states and state transition values), the new trust chain model is then established and that's the model of the user's own computer's trust chain. Our calculation and analysis are based on the new model. We make use of PRISM to make our algorithms and realization simple. An overview of the standard perspective of the tool can be seen in Figs. 5 and 6 as follows.

4 Conclusion and Future Work

The security of trust chain is very important in trusted computing, but there are no tools that can quantify it. Formal methods of modeling the trust chain and verifying its security properties have been put forward by many scholars, but these methods are hard to put into practice. We develop this tool to help users to figure out some security parameters, so as to get a clear understanding of the security situation of their trusted computers. For future work, we plan to integrate our recent work on trust chain security properties measurement, and composite our results to evaluate the security of trust chain, and we are also interested in extending our trust chain model to expand its applicability.

References

1. Xu, M., Zhang, H., Yan, F.: Testing on trust chain of trusted coputing platform based on labeled transition system. Chin. J. Comput. **32**(4), 635–645 (2009)
2. Fu, L., Wang, D., Kuang, J.: Conformance testing for trust chain of trusted computing platform based on finite state machine. J. Comput. Inf. Syst. **7**(8), 2717–2724 (2011)
3. Zhan, J., Zhang, H.: Automated testing of the trusted platform module. J. Comput. Res. Dev. **46**(11), 1839–1846 (2009)
4. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: probabilistic symbolic model checker. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS, vol. 2324, pp. 200–204. Springer, Heidelberg (2002)
5. Ching, W.-K., Huang, X., Ng, M.K., Siu, Tk: Markov Chains: Models, Algorithms and Applications, vol. 189. Springer, Heidelberg (2013)
6. Christel, B., Joost, P.K.: Principles of Model Checking, pp. 757–765. The MIT Press, Cambridge (2008)
7. CVSS. http://www.first.org/cvss
8. Bratus, S., D'Cunha, N., Sparks, E., Smith, S.W.: TOCTOU, traps, and trusted computing. In: Lipp, P., Sadeghi, A.-R., Koch, K.-M. (eds.) Trust 2008. LNCS, vol. 4968, pp. 14–32. Springer, Heidelberg (2008)
9. Bryan, J.P.: Trust extension as a mechanism for secure code execution on commodity computers. Ph.D.thesis, School of Electrical and Computer Engineering Carnegie Mellon University, April 2010

10. Zhen, H.D., Cong, T., Li, Z.: A decision procedure for propositional projection temporal logic with infinite models. Acta Informatica **45**(1), 43–78 (2008)
11. Tian, C., Duan, Z.: Model checking propositional projection temporal logic based on SPIN. In: Butler, M., Hinchey, M.G., Larrondo-Petrie, M.M. (eds.) ICFEM 2007. LNCS, vol. 4789, pp. 246–265. Springer, Heidelberg (2007)