Verification of an Integrated Role-Based Access Control Model

Saad Zafar[1], Kirsten Winter[2], Robert Colvin[3] and R. G. Dromey[4]

[1]Institute for Integrated and Intelligent Systems, Griffith University,
[4]Software Quality Institute, Griffith University,
4111, Brisbane (Nathan), Australia
[2,3]School of Information Technology and Electrical Engineering/ARC Centre for Complex Systems, University of Queensland,
4072 Brisbane (St. Lucia), Australia
{s.zafar, g.dromey}@griffith.edu.au, {robert,kirsten}@itee.uq.edu.au

Abstract. Role-based access control (RBAC) has been acknowledged as an effective mechanism for specifying and enforcing access control policies. However, it is not always clear how an RBAC policy can be systematically integrated into the system design so as to preserve the desired security requirements. In this paper we propose a representation and a process that supports systematic integration of an access control policy into the system design. The integrated design process uses a single notation, called Behavior Trees (BT), for specifying both the RBAC policy and the system design. Furthermore, the same notation is used for formal verification of both the RBAC policy and the system design. We also present strategies and challenges for automated translation of the extended BT notation required for specifying an RBAC policy.

Keywords: Access Control, Security Policy, RBAC Policy, Verification, Validation, Model Checking.

1 Introduction

Role-based access control (RBAC) is a mechanism for specifying and enforcing organizational access control policies [1]. Because of their flexibility and ease of use RBAC models are now viewed as a preferred “generalized approach to access control” [2]. RBAC models have been used in diverse applications ranging from web-based applications to the health-care industry. The renewed focus on RBAC has been attributed to the fact that it addresses many needs of the increasing web-based applications where availability and protection of data are major concerns [3]. In RBAC these concerns are addressed easily as most of the data availability and protection policies are generally aligned with the roles that individuals play within an organization [4].

A number of RBAC policy models and specification languages have been proposed in the literature [3]. The need for visualization of access control policies by using graphical notation has been highlighted in [5]. More recently, formal methods have been used for verification of RBAC specification. In [6], RBAC constraints have been expressed using LTL formalism. Others have proposed use of Alloy [7] and SPIN [8] for formulating RBAC security policies for formal verification. However, it is not always clear how the access control models integrate with application specifications. Also, it is often difficult to get a complete integrated view of the access control policy.

In comparison, BT notation offers a single notation, with formal semantics, for specification of security policies and the specification of system design [9]. This not only facilitates an integrated and incremental development process but also reduces the accidental complexity of the overall design process [10]. The formal semantics of BT notation allows for automated verification of the BT specification model [11]. The integrated view of the system design generated by the BT specification also facilitates combining and resolving conflicts in security policies. To specify RBAC policies, where the roles a user may play are determined by its membership in the access control sets, we have extended the BT notation to include sets and set operators. The syntax has also been incorporated into the existing operational semantics and the translation semantics for automated translation of BT to SAL specification. The SAL specification can then be used to verify the *safety* properties [12] of the access control model.

In related unpublished work, an operational semantics for the BT notation has been developed and implemented as a simulation tool. The simulator can be used for validation, and can also be used to check simple safety properties of the model. During the verification process, the simulator can be used for observing any counter examples generated by the model checker.

The rest of the paper is organized as follows. Section 2 gives a brief background on RBAC and BT notation. Section 3, introduces the proposed BT-based RBAC model which we refer to as BT-RBAC. In section 4, we illustrate the strategies we plan to use for formal verification of BT-RBAC specification. In section 5, we argue the need and suitability of the BT based RBAC model and section 6 provides a summary of future work.

2 Background

2.1 Role-Based Access Control

The central concept behind RBAC is that access control permissions are assigned to roles and users are then assigned to suitable roles [4]. As roles have more permanence in an organizational context than users, assigning permissions to roles greatly facilitates access control management. Individual users can be assigned and de-assigned from roles without having to manage access control permissions for each individual separately.

The NIST RBAC model [2] defines a core RBAC model as a collection of elements, element sets, and relations to specify a role-based access control system.

The basic data elements in the core RBAC are users, roles, permissions, operations, and objects. A user can be a human being, a machine or an intelligent autonomous agent. A role corresponds to the job function in the context of an organization which has authority and responsibility associated with it. Permission is the authority to perform an operation on protected objects within a system. An operation corresponds to application-specific user function. An object in an RBAC model may contain or receive some information. The model also defines a session during which a user can activate some of the roles that have been assigned to him or her. In addition, the model defines a hierarchical RBAC which facilitates role management by allowing inheritance of permissions and users among roles. The constrained RBAC specifies separation of duty constraints to avoid any conflict of interest in a policy.

2.2 The Behavior Tree Notation

The BT-based design process involves translating individual requirements into the BT notation, referred to as *Requirements Behavior Tree* (RBT), one at a time [9]. The RBT for each of the requirements is then composed together into an *Integrated Behavior Tree* (IBT) to provide an integrated view of the system requirements. The IBT is systematically refined into a *Design Behavior Tree* (DBT) by taking a number of design decisions. The impact of these design decision is readily visualized as the changes are applied to an integrated view of the requirements, thereby, potentially reducing the complexity of the design process [13]. It is also possible to derive the system architecture from the DBT which is presented in the form of *Component Interaction Network* (CIN) diagram. The behavior of individual components may also be projected out using *Component Behavior Tree* (CBT).

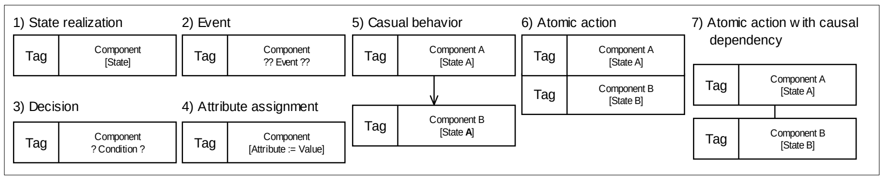


Fig. 1. A summary of BT syntax

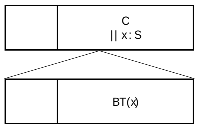
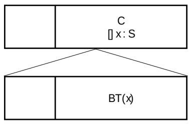
A Behavior Tree node is used to specify a component and its state. It is also used to express conditions, events, attribute assignment and data-in and data-out behaviours of a component. Two BT nodes joined together with an arrowed line represent causal behaviour and the direction of flow of control. Two nodes joined together with a line without an arrow specify an atomic action with causal dependency. Atomic actions without causal dependencies are specified as BT nodes boxed together without a joining line between them. A summary of BT syntax is presented in Fig. 1.

The Behavior Tree framework has been evaluated as a means of early defect detection in requirements [14]. It has also been proposed as an effective means for managing requirements change in the latter stages of the design process [15]. The suitability of the notation has been evaluated in the design and development of embedded systems [13] and safety critical systems [11]. The integration of security requirements into design of a safety and security critical system has been illustrated in [16]. In [17] IEEE 802.11i WLAN security protocol has been modelled and verified using the BT notation. Our current research involves evaluating application of BTs in the domain of security engineering.

2.3 Modelling Role-Based Access Control using BT

To specify RBAC policies, the original BT syntax has been extended to include set and set operations to model role-based access control requirements. A set can be specified as a component itself or can be an attribute of a component. Basic set operations like union, compliment, and intersection are allowed, as well as membership and cardinality. In addition, two other constructs have been introduced: one for selection of all elements used in parallel execution of a BT segment, and another one for selection of one element (chosen non-deterministically) from a given set (Table 1). The static analysis of the RBAC policy is possible by automatically translating DBT into the SAL specification language. SAL is an integrated environment of static analysis tools that include tools for model checking and theorem proving [18]. A formal specification of the translation rules for a core of the BT notation into the SAL specification is provided in [19].

Table 1. BT set notation and its semantics.

Component-State	Semantics	Component-State	Semantics
C [S := {}]	Set S realizes a state where S becomes empty.	C ? x: S ?	Is x a member of set S?
 C || x: S; BT(x)	Parallel execution of Behavior Tree BT for every element of S.	 C [] x: S; BT(x)	Execution of Behavior Tree BT for one element x in set S. x is non-deterministically selected from set S.
C ? |S| < *n* ?	Set cardinality: Is cardinality of set S less than n?	C [S := S + A]	Union of set S with set A.
C [S := S + {x}]	Adding an element x to set S.	C [S := S - A]	Set difference. Set S minus set A.
C [S := S - {x}]	Subtraction of element x from set S.	C [S := S >< A]	Set intersection: Intersection of set S and set A.

3 A Component-state Based RBAC Model

In a BT-based RBAC specification, roles are represented as sets. A session is represented as a component and may contain active roles as attributes. Objects are represented as components and allowable operations on them are represented as its states. All user actions are modelled as input events. An operation on object is an input event, which causes the object component's state to be changed (Fig. 2). We have used a simple case study as a running example to illustrate how the RBAC policy can be specified using BT notation. The case study is a simplified version of an online-classroom system that requires a decentralized enforcement of RBAC policy in a collaborative environment [8]. The security requirements that must be preserved in the online-classroom system are discussed in section 4.

3.1 Role Specification

A role is specified as a set in BT-RBAC. The authorized members for a role are added or removed using the set operations. The membership of a user in a specified role is checked by using the set membership function in BT. The count of members in a role is the cardinality of the set representing the role. In Fig. 2; (a) a user Admin first creates a role R-Student, (b) a user *convener* assigns a user *S1* to the role R-Student, (c) a user *convener* removes the user *S1* from the role *R-Student*.

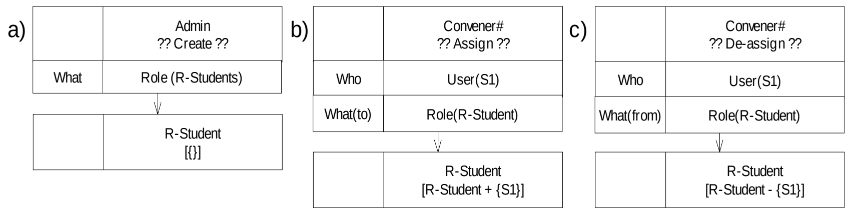


Fig. 2. a) A user *Admin* creates a role *R-Student*. b) A user *Convener#* assigns a member *S1* to role *R-Student*. c) A user *Convener#* deassigns user *S1* from role *R-Student*.

3.2 Role Membership Constraints

A role-cardinality constraint in an RBAC policy specifies the maximum number of members allowed in a particular role [1]. In BT-RBAC model, the role membership constraints are enforced by specifying them at the time of assigning members to a role. For instance, a role-cardinality constraint specifies the maximum number of members allowed in a role. Fig. 3(a) illustrates how the role-cardinality constraint may be enforced. A member *A1* can only be assigned to *R-Assistant* role if the cardinality of the role is less than 2. It may be noted that BT nodes are atomically composed to avoid the possibility of set cardinality being changed by any interleaving threads.

3.3 Separation of Duty

The concept of separation of duty has been prevalent in many commercial systems as one of the mechanisms for prevention of fraud and controlling an error in a business process [20]. The basic concept revolves around the idea that a business operation is divided into many sub-tasks and then ensuring that each of these tasks is performed by different persons, thereby, reducing the risk of fraud and error in execution of a critical business operation. The concept of separation of duty may be divided into two broad categories; static separation of duty and dynamic separation of duty.

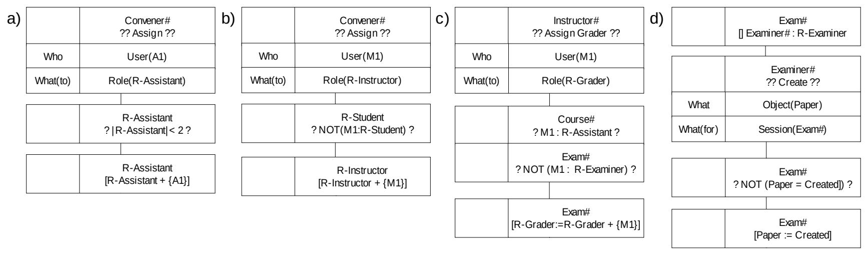


Fig. 3. Examples of (a) role membership constraint; (b) static separation of duty; (c) dynamic separation of duty; and (d) object-based separation of duty.

In static separation of duty (SSD) policy two roles are made mutually exclusive by disallowing membership of one person in both the roles [21]. The policy avoids conflict of interest by disallowing one user to be a member of a second conflicting role. In Fig. 3(b), a student may not be allowed to join the role of an instructor as a part of the online-classroom security policy. In dynamic separation of duty (DSD) policy, a user is allowed to perform conflicting tasks that would not be allowed under a SSD policy as long as the policy minimizes the chances of fraud and error [21]. A typical DSD policy requires that two restricted roles may have common members but they may not join both the roles at the same time [21]. For instance, any member from R-Assistant can join either R-Examiner or R-Grader roles but one member cannot join both the roles at the same time (Fig. 3.c).

In another variation of the DSD, object-based separation of duty (OBSD), users are not allowed to act upon a single object twice [21]. In figure 3(d), the members of R-Examiner role are not allowed to create an exam paper if one has already been created. The integrated policy for the online-classroom is presented in Fig. 4. The figure is only intended to give the reader a feel of the size and structure of the integrated policy specification. The other variations of DSD which have been modeled in the final specification include operational separation of duty (OSD), and order-dependent/independent history-based separation of duty (HSD) [21].

4 Validation and Verification of BT-RBAC Model

The automated translation of BT into SAL specification language makes it possible to model check the design Behavior Tree. A prototype BT simulator has also been developed to support early validation of the BT specification. The strategies and challenges in developing extensions to the existing tools to support validation and verification of BT-RBAC are discussed in this section.

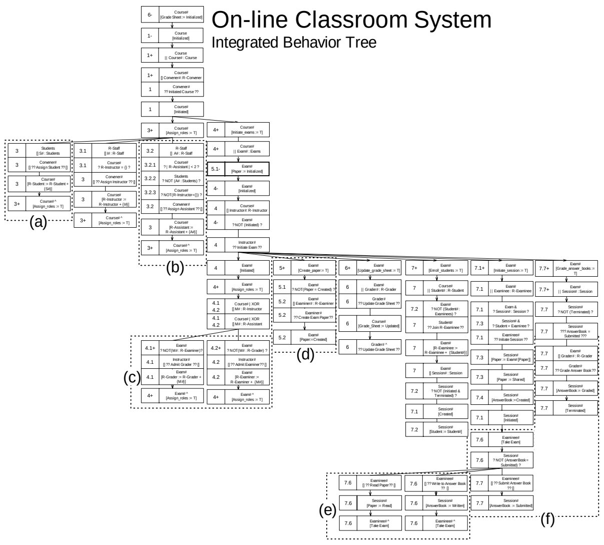


Fig. 4. Integrated RBAC Policy (a) *Role membership assignment*: A member from R-Convener role assigns members to R-Student role. (b) *Role membership constraints and static separation of duty*: A member from R-Staff role can only be assigned to R-Assistant role if he/she is not a member of R-Student, if the R-Assistant role cardinality is less than two and if R-Instructor role has been assigned a role. (c) *Dynamic separation of duty:* A member from R-Assistant is assigned to either R-Examiner or R-Grader role but can not be assigned to both the roles at the same time. (d) *Object-based separation of duty:* The members of R-Examiner role are not permitted to create an exam paper twice. (e) *Order-independent history-based separation of duty:* An examinee can read from the exam paper and write to an answer book before it is submitted. (f) *Order-dependant history-based separation of duty:* A member from R-Grader role can only grade the answer book if it has been submitted.

4.1 BT to SAL Translation

To include the new features of the BT notation for manipulating sets, we have had to extend the given translation rules [19]. For the most common set theoretic operations (like adding and removing a member from a set, union and intersection of sets, membership and empty set queries) we follow the suggestions in [22] where a translation from the Z set-theoretical specification language into the SAL language is introduced. For the encoding we can utilize a predefined *set context* of the SAL package. This context defines a type *SET* and all the necessary set operators.

The set context also provides an operator for the cardinality of a set. However, the encoding of this operator in SAL is known to be computationally expensive. Where possible the use of the cardinality operator should be avoided or circumvented, e.g. by auxiliary counters.

As a challenge we encounter two new constructs that diverge from basic set-theoretic operators:

a) For every member *c*, written (||c:S;T(c)), execute subtree *T(c)*.

b) Non-deterministically choose a member *c*, written ([]c:S;T(c)), execute subtree *T(c)*.

To translate these two constructs we unfold the tree to its maximal size, based on predetermined maximal sizes for sets *S*. For instance, if *Colour* is a type defined as {red, blue, yellow}, and *S* is declared as a set of *Colour*, then both of the constructs are unfolded into three branches, corresponding to each element of *Colour*. Because *S* may change dynamically during run time, i.e., contain some subset of the three colours, we only enable the threads which correspond to values that are currently in the set. The others remain disabled. In case of (a) above, all activated threads run concurrently, whereas in case of (b) only one activated thread is chosen and all others become disabled.

For this purpose the user needs to input the maximal size of the sets that are to be unfolded. Moreover, due to the fact the model checking approach used here works on finite systems only, all sets have to be finite. In most cases, however, it is sufficient to use a model with only small sets in order to show violations of safety and security requirements.

4.2 Verification of BT-RBAC Properties using Model-Checker

The safety property of an access control configuration refers to inaccessibility of protected objects to unauthorized users [12]. Since the safety property is said to be *undecidable*, the access control models often use constraints on access control to facilitate safety analysis of the model [5]. As discussed earlier, in the BT-RBAC model constraint expressions are placed on the access control to facilitate analysis of the model. In our running example we follow the approach discussed in [8] to verify that: 1) a participant of the examinee role cannot access the content of the question paper before the start of his/her own exam, and 2) a participant of the examinee role can only modify his/her answer book before end of his/her exam-session.

The first property embodies confidentiality or information flow policy. A confidentiality policy aims to protect unauthorized disclosure of information. In the example, the constraints placed on the access of the exam paper restrict its access before the start of student's own exam session. The verification of the property involves the verification of these constraints on the flow of information. The following LTL formula (1) is used to verify the first property. The actual SAL syntax has been slightly changed in the formula to improve readability. The *U* operator in the formula is read as "*p* until *q*", i.e. *p* holds until a state is reached where *q* holds.

```
(FORALL(s:SessionType):                                          (1)
    (contains?(S,s)) AND NOT(s.Paper=read)
      U  (s.Status=initiated OR NOT(contains?(S,s)));
```

We benefit from the SAL's advanced input language that allows us to use record types and set operators. The record types used in Formula 1 have been defined as follows.

```
StatusType : TYPE={initiated, terminated};
   PaperType : TYPE={read,unread};
   AnswerBookType : TYPE={not created, created,
submitted, graded};
SessionType : TYPE = [# Status : StatusType,
                        Paper :  PaperType,
                        AnswerBook : AnswerBookType #];
Session : TYPE=set{SessionType}!Set;
% The variable S for being a set of sessions:
S : SessionType
```

The second property relates to the *integrity* policy of the RBAC model. To verify the integrity of an RBAC model we verify that access rights are not leaked during role assignments [8]. In this case, we verify that the ability to the answer book is not leaked after the session has been terminated. The following LTL formula (2) is used to verify the second property. The *G* operator in the formula is read as "always *p*", i.e. *p* is always true.

```
(FORALL(s:SessionType):                                                    (2)
      G(s.Status=terminated AND contains?(S,s)
               =>  s.AnswerBook=submitted);
```

5 Discussion

In this paper we have presented the BT-RBAC model as a proof of concept for a completed model. The key potential benefits of the model- are discussed here.

The BT-based RBAC model is flexible and allows better integration with systems requirements. It can potentially be used to specify *fine-grained-and-coarse-grained specifications* [23] that allows for specific access control rules for users and objects in an RBAC configuration. This is possible since the Behavior Tree notation supports modelling of complex and concurrent system behaviour. In our example, we were able to model a *decentralized* [24] computer supported cooperative work system with relative ease. A completed BT-RBAC template would support repeatability and accuracy of specification.

Another requirement of an access control model is ensuring the reliability of the input [23]. In the BT specification this is ensured by putting the necessary checks with every input event as shown in our running example. These checks may include authentication of users along with *conditional authorization* [23] like time constraints on roles. The *principle of least privilege* [23] is supported by constraining user behaviour to the assigned roles and by performing necessary checks before every user action. To this end various *separation of duty* [23] policies are often used. The BT notation is flexible enough to model most of these policies. *Policy combination and conflict resolution* [23] is inherently supported in BT specification. An Integrated Behavior Tree assists in resolving conflicts in requirements and helps identify any incompleteness in them [25]. The integrated view of an access control policy can also play the same role in identification of conflicts and incompleteness in the policy. Policy combination is achieved in a manner similar to the integration of requirements behaviour trees into a single IBT [25].

Integration of security requirements with systems engineering is essential for developing security critical applications [26]. The security requirements of a system must be integrated with the rest of the system requirements to achieve this objective. The integration requirements must then be validated and verified for correctness and completeness. The isolated verification and validation of security policies may lead to false sense of faith in the security policy. As illustrated in [16], the BT-based design process offers a platform for integration of security requirements into the design of a system. The incremental proof of a system is possible by first verifying the correctness of the policy and then verifying the integrated system specification that incorporates the policy.

The BT-based design process supports accurate formalization in the early requirements engineering phase by providing a systematic translation technique that translates informal requirements into BT notation. The BT editing tool helps traverse through each individual requirement during the translation process. The components and their corresponding, states, events, or conditions are identified using the tool. Any ambiguities, aliases and incompleteness is resolved and documented for validation. The tool automatically maintains traceability by marking each BT node with the corresponding requirement number. The requirements specification is then refined into a design specification in the form of a DBT. The key advantage of the approach is that the requirements specification notation is same as the *design notation* [27]. The use of single language through out the development process is aimed at facilitating the integrated design process and reducing the *accidental complexity* of the process [10]. The formal semantics of the BT notation makes it possible to formally verify and simulate the specification. In addition, the graphical tree structure of the notation is intended to enhance the comprehensibility of the specification [25].

The early validation and verification of BT-RBAC is possible by extending the operational and translational semantics of the BT to include BT set notation. The specified policy can be validated by generating all the traces of the finite system (subject to hardware constraints) and check the safety properties on the state of the system at all intermediate steps. A potentially infinite (reactive) system can be made finite by limiting the number of entities participating in a system and restricting the number of interactions with the environment. The BT-RBAC policy can also be verified using the tools in SAL environment by specifying security requirements as LTL formulas.

6 Future Work

The BT-RBAC model presented here is envisioned to have application-independent templates necessary to specify most common RBAC policies. The model will be extended to include identification and authentication templates for better integration of security requirements. The possibility of automatically generating various views like role permissions, and access control lists from the specification will also be explored. We will also investigate the performance related issues in simulating and model checking of the integrated BT-Models. To meet our research objectives of developing a systematic approach for designing dependable systems the use of BT design process for integrating safety and security requirements are also being investigated.

Acknowledgments. This work is partially funded by Australian Research Council (ARC) under the ARC Centres of Excellence program. We would also like to acknowledge the contribution of our colleagues at the Building Dependability into Complex Computer-Based Systems research sub-program established under the ARC Centre for Complex Systems.

References

1. Ferraiolo, D.F., J. Cugini, and D.R. Kuhn, *Role-Based Access Control (RBAC): Features and Motivations.* Proceedings, 11th Annual Computer Security Applications Conference, 1995: p. 241-48.
2. Ferraiolo, D.F., et al., *Proposed NIST Standard for Role-Based Access Control.* ACM Transactions on Information and System Security, 2001. **4**(3): p. 224-274.
3. Ferrari, E., *Guest editorial: Special issue on access control models and technologies.* ACM Trans. Inf. Syst. Secur., 2005. **8**(4): p. 349-350.
4. Sandhu, R.S., et al., *Role-Based Access Control Models.* IEEE Computer, 1996. **29**(2): p. 38-47.
5. Jaeger, T. and J.E. Tidswell, *Practical Safety in Flexible Access Control Models.* ACM Transactions on Information and System Security, 2001. **4**(2): p. 158-190.
6. Drouineaud, M., M. Bortin, and P.S. Torrini, K. *A first step towards formal verification of security policy properties for RBAC.* in *Proceedings Fourth International Conference on Quality Software, 2004. QSIC 2004.* 2004.
7. Hughes, G. and T. Bultan, *Automated Verification of Access Control Policies*. 2004, Computer Science Department, University of California, Santa Barbara, CA 93106, USA.
8. Ahmed, T. and A.R. Tripathi, *Static verification of security requirements in role based CSCW systems.* Proceedings of the eighth ACM symposium on Access control models and technologies, 2003: p. 196-203.
9. Dromey, R.G. *From Requirements to Design: Formalizing the Key Steps*. in *Proceeding, First International Conference on IEEE International Conference on Software Engineering and Formal Methods (SEFM 2003)*. 2003. Brisbane: IEEE Computer Society.
10. Dromey, R.G., *Climbing over the 'No silver bullet' brick wall.* IEEE Software, 2006.
11. Grunske, L., et al. *An Automated Failure Mode and Effect Analysis based on High-Level Design Specification with Behavior Trees*. in *Fifth International Conference on Integrated Formal Methods (IFM2005)*. 2005. Eindhoven, The Netherlands.
12. Harrison, M.A., W.L. Ruzzo, and J.D. Ullman, *Protection in operating systems.* Commun. ACM, 1976. **19**(8): p. 461-471.
13. Zafar, S. and R.G. Dromey. *Managing Complexity in Modelling Embedded Systems*. in *Systems Engineering / Test and Evaluation Conference SETE2005*. 2005. Brisbane, Australia.
14. Dromey, R.G. and D. Powell, *Early requirements defects detection.* TickIT Journal, 2005. **4Q05**: p. 3-13.
15. Wen, L. and R.G. Dromey, *From Requirement Change to Design Change.* Proceedings, IEEE International Conference on Software Engineering and Formal Methods, 2004: p. 104-113.

16. Zafar, S. and R.G. Dromey. *Integrating Safety and Security Requirements into Design of an Embedded System*. in *Asia-Pacific Software Engineering Conference*. 2005. Taipei, Taiwan: IEEE Computer Society.
17. Sithirasenan, E., S. Zafar, and V. Muthukkumarasamy. *Formal Verification of the IEEE 802.11i WLAN Security Protocol*. in *Australian Software Engineering Conference (ASWEC '06)*. 2006. Sydney, Australia.
18. Shankar, N. *Combining Theorem Proving and Model Checking through Symbolic Analysis*. in *CONCUR'00: Concurrency Theory*. 2000: Springer-Verlag.
19. Grunske, L., K. Winter, and N. Yatapanage, *Defining and Parsing the Abstract Syntax of Visual Languages with Advanced Graph Grammars - A Case Study Based on Behavior Trees.* To Appear, 2006.
20. Clark, D.D. and D.R. Wilson, *A Comparison of Commercial and Military Computer Security Policies.* IEEE Symposium on Security and Privacy, 1987: p. 184-194.
21. Simon, R.T. and M.E. Zurko, *Separation of Duty in Role-based Environments.* Proceedings: 10th Computer Security Foundations Workshop, 1997: p. 183-194.
22. Smith, G. and L. Wildman. *Model checking Z specifications using SAL*. in *4th International Conference of B and Z Users (ZB 2005)*. 2005. Guildford, UK: Lecture Notes in Computer Science, Springer-Verlag.
23. Vimercati, S.D.C.d., S. Paraboschi, and P. Samarati, *Access control: principles and solutions.* Softw. Pract. Exper., 2003. **33**(5): p. 397-421.
24. Bhatti, R., et al., *X-GTRBAC Admin: A Decentralized Administration Model for Enterprise-Wide Access Control.* ACM Transactions on Information and System Security, 2005. **8**(4): p. 388-423.
25. Dromey, R.G., *Genetic Design: Amplifying Our Ability to Deal With Requirements Complexity.* Lecture Notes in Computer Science, 2005. **3466**: p. 95-108.
26. Devanbu, P.T. and S.G. Stubblebine. *Software Engineering for Security: A Roadmap*. in *International Conference on Software Engineering (ICSE 2000) - Future of SE Track*. 2000.
27. Hall, A. *Software Verification and Software Engineering*. in *(Keynote Address) Verified Software: Theories, Tools, Experiments (VSTTE'05)*. 2005. Zürich.