Early Validation and Verification of a Distributed Role-Based Access Control Model

Saad Zafar[1,4], Robert Colvin[3,4], Kirsten Winter[3,4], Nisansala Yatapanage[2,4], and R. G. Dromey[2,4]
[1]*Institute for Integrated and Intelligent Systems,*
[2]*Software Quality Institute, Griffith University, Australia*
[3]*School of Information Technology and Electrical Engineering, University of Queensland*
[4]*ARC Centre for Complex Systems (ACCS), Australia [website:http://www.accs.edu.au]*
{s.zafar,n.yatapanage,g.dromey}@griffith.edu.au, {robert, kirsten}@itee.uq.edu.au

Abstract

To ensure correct implementation of complex access control requirements, it is important that the validated and verified requirements are effectively integrated with the rest of the system. It is also important that the system can be validated and verified early in the development process. In this paper we present an integrated, role-based access control model. The model is based on the graphical Behavior Tree notation, and can be validated by simulation, as well as verified using a model checker. Using this model, access control requirements can be integrated with the rest of the system from the outset, because: a single notation is used to express both access control and functional requirements; a systematic and incremental approach to constructing a formal Behavior Tree specification can be adopted; and the specification can be simulated and model checked. The effectiveness of the model is evaluated using a case study with distributed access control requirements.

1. Introduction

Due to increasing complexity of data availability and protection requirements, many modern systems incorporate a complex set of access control policies. The de facto approach to modeling these access control requirements is Role-based access control (RBAC), because it can greatly facilitate access management in a flexible manner [1]. The flexibility and ease of use in RBAC comes from the fact that most data protection and availability policies are aligned to the roles that individuals play within an organization; access management is then a matter of assigning roles to individuals, rather than managing the control rights for each individual separately. In recent years a number of RBAC models have been proposed to facilitate specification and analysis of access control requirements [2].

Traditionally, security engineering has been performed in isolation from systems engineering [3]. This often results in late integration of security requirements into system design in a post-hoc manner. In this paper we propose extending an existing requirements engineering notation, Behavior Trees, with language constructs based on the NIST standard for RBAC [1].

The Behavior Tree (BT) notation is a graphical notation [4], which is used in a development approach called Behavior Engineering (BE). The BE approach provides systematic translation of informal requirements into a formal representation [5]. An integrated view of the requirements is generated in the form of an Integrated Behavior Tree (IBT), which provides a platform for requirements analysis and design. With tool support the BT model can be simulated and model-checked for correctness. The approach has proved successful in industrial practice [6].

By adding RBAC constructs to the BT notation we extend its capabilities of handling systems with security requirements. In addition, due to the nature of the notation and the Behavior Engineering process, we are able to avoid many of the problems associated with incorporating security requirements into a system design. Existing highly specialized RBAC models often use a different notation to the design notation making the integration process difficult [7]. Furthermore, the informal and semi-formal RBAC models may not be completely verified for their correctness, which may compromise the security of the final system. On the other hand, the formally specified models may not be easy to use and validate due to their mathematical nature [8]. The trade off for the users is

1530-1362/07 $25.00 © 2007 IEEE
DOI 10.1109/ASPEC.2007.20

often between usability and formal tractability being offered by the different models.

These problems are avoided in the BT-RBAC because it is an integrated model that aims to simplify formal specification, validation, verification and integration of access control requirements in the system design. The requirements translation process is augmented with an access control specification template which facilitates the requirements specification process. Conflict resolution is supported during the requirements integration process when an IBT for the RBAC is generated. The model can be validated via simulation of the IBT and can be model checked using the automated translation into the input language of the SAL model-checker [9], [10]. The model's integration with functional requirements is assisted through the use of a single notation for all types of requirements. We evaluate the effectiveness of the model using a simple case study with distributed access control requirements.

The paper is structured as follows. Section 2 gives a brief background on RBAC and Behavior Engineering. Section 3 introduces the BT-RBAC model. We show how distributed RBAC requirements can be modeled using a case study as a running example. Section 4 describes tool support for simulation and model-checking. In Section 5 we review related work and Section 6 presents conclusions and future work.

2. Background

2.1. Role-based access control

The access control mechanisms in computer systems define the permission that users or processes have to access protected resources in a computer system. In a role-based access control (RBAC) mechanism, these rights are defined based on the role that individuals are assigned to in an organization. The advantage of assigning access rights to roles instead of individual users is that roles have more permanence in an organizational context than users [11]. Individual users can be assigned and de-assigned from roles without having to manage the access control permissions for each individual separately.

The NIST RBAC model [1] defines sets of basic elements and relations that are needed to specify access control requirements. The basic 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. An *operation* corresponds to an application-specific user function. An *object* in a RBAC model may contain or receive information. *Permission* is the authority to perform an operation on protected objects within a system. The model also defines a *session* during which a user can activate some of the roles that have been assigned to him or her. The constrained RBAC specifies separation of duty constraints to avoid any conflict of interests in a policy.

2.2. Behavior Engineering

The Behavior Engineering approach uses a graphical notation, called Behavior Trees (BTs), to formalize and integrate all system requirements in a single tree structure [12]. The approach includes a rigorous requirements translation process that translates one informal requirement at a time into a formal representation. The individual BTs, representing each requirement, are then composed together into a single tree-structure called an Integrated Behavior Tree (IBT). Early defect detection is supported as defects are tagged and resolved during the translation process [13]. A BT node is marked with a '+' sign if the associated behavior is implied and a '-' sign to mark behavior that is missing in the requirements. Requirements traceability is maintained by marking each BT node with the corresponding requirement number. Any integration defects need to be resolved once all the requirements are composed together. If a requirement does not integrate with other requirements it indicates an incompleteness problem or that the requirement does not form part of the system. These defects need to be resolved before an integrated view of the requirements can be finalized.

The use of a single notation throughout the process facilitates an integrated approach to system design and helps avoid accidental complexity that may be introduced when a number of different representations are used during different development phases [14]. Some of the recent research work based on the BT framework include [6, 15, 16].

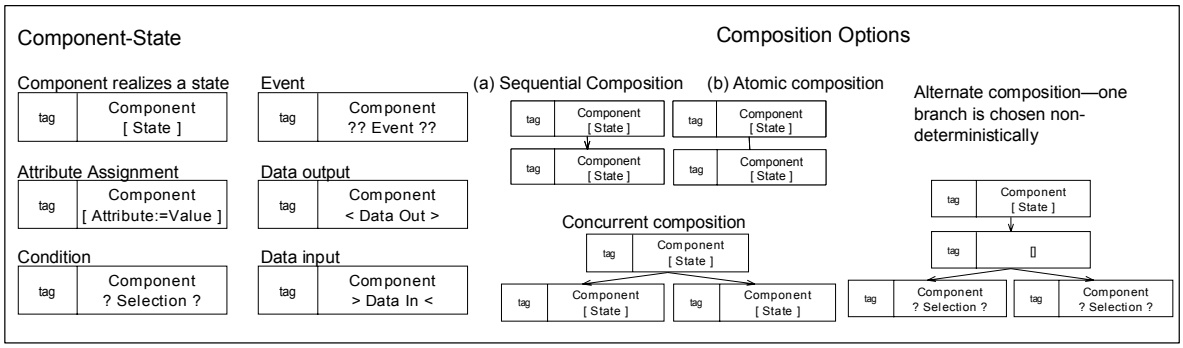

Figure 1: A summary of core BT syntax

The core elements of the BT notation are given in Fig. 1. A BT node is used to specify a component and its state. It is also used to express conditions, events, attribute assignment and data flow. Two BT nodes joined together with an arrowed line represents a causal behavior and the direction of flow of control. Two nodes joined together with a line without an arrow specify an atomic composition, i.e., no interleaving of actions from parallel processes is allowed. The parallel branches in the BT represent concurrent behavior by default and represent alternate branching when marked with the choice operator '[]', in which case only one of the branches is (non-deterministically) chosen to execute.

RBAC models include sets of users and objects, and therefore the BT notation has been extended with set operators such as '+', '-' and '><' for set union, difference and intersection, respectively. We write |S| for the cardinality of set S, and x:S for membership of element x in set S. In addition, two constructs have been introduced: one which models execution of a tree BT(x) for all members x of a set S, and one which models execution of BT(x) for some member x of set S. These constructs are referred to as *forall* and *forone,* respectively – see Fig. 2.

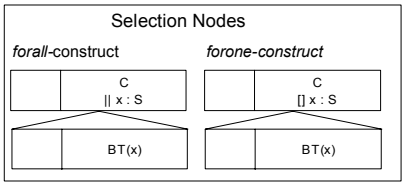

Figure 2: *forall* and *forone* constructs.

2.3. The case study

We now present a simple case study that will be used to evaluate the effectiveness of the BT-RBAC model. The case study, presented in [17], is that of an online-classroom system with distributed access control requirements. In a course activity a member of the *Conveners* role can initiate a *course* session. A *course* session contains three roles: *Instructors, Assistants,* and *CStudents*. Only one member from the *Staff* role can be assigned to the *Instructors* role. A member of the *Staff* role can be assigned to the *Assistants* role if (i) the role membership count is less than two, (ii) the member being assigned to the role is not a member of the *Students* role, and (iii) a member has already been assigned to the *Instructors* role.

The instructor of a course can initiate an *exam* activity for that course. In the *exam* activity, the instructor can admit members to the *Graders* and *Examiners* roles. Members of these roles can be selected from either the *Instructors* or the *Assistants* role of the course. A member of the *Examiners* role can create an exam *paper* if one has not been created before. A participant of the *Graders* role can update grades in the *grade sheet* for the course. Students enrolled in the course, i.e., members of the set *CStudents,* can join the *Examinees* role in the *exam* activity. An examinee can then initiate an *exam session* but not more than one. An examinee can read the exam *paper* and write to the *answer book* during the *exam session*. The grader can grade the *answer book* only after it has been submitted by the examinee. The session terminates when the member of the *Graders* role has finished grading the *answer book* for a session.

The following features of RBAC have been enforced in the case study: A *role-cardinality constraint* requires that the capacity of the role should not be increased by the addition of another member [18]. In the case study, at most one member can be assigned to the Instructors role (so an instructor can also be a student) (1). A maximum of two members can be assigned to the Assistants role (2). These requirements are formalized in the following formulas in linear temporal logic (LTL). **G**, **F** and **U** are temporal operators in LTL [19] which have the following meaning: the **G**(*p*) operator states that *p* is always true; **F**(*p*) means that *p* will eventually be true; and (*p* **U** *q*) states that *q* will hold eventually and *p* holds until *q* holds.

In a *separation of duty* constraint, a business process is divided into many sub-tasks and then these sub-tasks are assigned to different persons to ensure that no single person is responsible for performing a critical business operation [20]. The concept of separation of duty may be divided into two broad categories; static separation of duty and dynamic separation of duty. In *static separation of duty* (SSD) policies, two roles are made mutually exclusive by disallowing membership of one user in both roles [21]. The SSD requirement in the classroom case study states that a member of the Students role of a course cannot be assigned to the Assistants role of that course (3).

A *dynamic separation of duty* (DSD) policy requires that two restricted roles may have common members but they may not join both the roles at the same time [21]. A DSD policy may have other variations [21]. In an *object-based separation of duty* (OBSD), users are not allowed to act upon a single object twice. For example, a student cannot initiate another session if one has already been initiated for a particular exam (4). *History-based separation of duty* is a more flexible variation of DSD in which a user is not only allowed to perform more than one action on an object but may also be allowed to perform all the actions in a business task as long as necessary safeguards are implemented in the policy. For example, a member from the Graders role can only grade the answer book if it has been submitted (5); an examinee from the Examinees role can read from the exam paper and write to the answer book as long as it has not been submitted (6); and a member of the Examinees role cannot access the content of the question paper before his/her exam session has been initiated (7).

$$\forall c : Courses \bullet \mathbf{G}\ (size(c.Instructors) \leq 1) \quad (1)$$

$$\forall c : Courses \bullet \mathbf{G}\ (size(c.Assistants) \leq 2) \quad (2)$$

$$\forall c{:}Courses \bullet \mathbf{G}(c.CStudents \cap c.Assistants{=}\varnothing) \quad (3)$$

$$\forall u : Users,\ \forall s_1, s_2 : Sessions \mid s_1 \neq s_2 \wedge s_1.exam\_id = s_2.exam\_id \bullet \mathbf{G}\ ((s_1.status = initiated \wedge s_1.student = u) \Rightarrow \neg(s_2.status = initiated \wedge s_2.student = u)) \quad (4)$$

$$\forall s : Sessions \bullet \mathbf{G}\ (s.answerBook \neq submitted) \vee (s.answerBook \neq graded\ \mathbf{U}\ s.answerBook = submitted) \quad (5)$$

$$\forall s : Sessions \bullet \mathbf{G}\ ((s.answerBook = submitted) \Rightarrow \neg\ \mathbf{F}\ (s.paper = read \vee s.answerBook = written)) \quad (6)$$

$$\forall e : Exams,\ \forall s : Sessions \mid s.exam\_id = e.id \bullet \mathbf{G}\ (s.paper \neq read\ \mathbf{U}\ s.status = initiated) \quad (7)$$

3. The BT-RBAC model

The BT-RBAC is an integrated model that aims to simplify the formal specification, validation, verification and integration of access control requirements into the system design. All the basic elements in the model, which have been derived from the NIST RBAC standard [1], are specified using the component-state model of the BT notation. To specify access control requirements, the BT nodes representing different components of the model are composed together for each requirement using the BT-RBAC template (Figure 3a). These requirements are integrated together in the form of an IBT to generate an integrated view of the complete policy.

The objects, sessions, and/or roles are first selected using *selection nodes* (see Figure 2). This is followed by a *permission node* that specifies the member action associated with the permission being granted. Semantically, the permission node is a BT event type, which is associated with the component specified in the node. The permission node also specifies the relationship between the action and other components that are involved in the permission being granted (see Figure 3(a)). The permission node is then followed (optionally) by *constraint nodes*. These are BT selection nodes and can be used to specify constraints on role membership, session operations or object operations. Lastly, the *operation node* specifies the operation on the object or the session component as a BT state-realization or attribute-assignment node. It may be noted that the permission, constraint and operation nodes are atomically composed to avoid side effects from any interleaving threads. The use of the model is explained using the first few requirements of the online-classroom system given in Table 1.

Table 1: Access control requirements

No	Requirement
R1	A member from Conveners role in a course can initiate a course activity.
R2	A member from Conveners role in a course can assign a member from Staff role to course's Assistants role. A maximum of two assistants can be assigned to Assistants role. A student cannot be assigned to Assistants role. An Assistant is only assigned to her role if Instructors role is not empty.
R3	A member from Conveners role in a course can assign only one member from Staff role to course's Instructors role.
R4	A member from Conveners role in a course can assign only members from Students role to course's CStudents role.

Requirement *R1* in Table 1 is formalized in Figure 3(b). First a *Course* session is selected from the *Courses* set, using the *forall* BT construct. Next a *Convener* is selected from the course's *Conveners* role

using the same construct. Both the *Course* and the *Convener* represent all the members in their respective sets. The permission node specifies that when the *Convener* component exercises the permission granted to the role the subsequent nodes in the tree are executed. The permission granted to the *Convener* is that he/she can initiate a course. For the verb used in the permission node (initiate) we can ask *what, where, when, who* and *how* questions to identify all the objects associated with the verb. These questions can help guide the elicitation and ensure completeness of the permission being specified. In this case we may ask the question, "Convener assigns what?". The answer is, *Session(Course)*. There are no constraints specified with this requirement. When the operation node is executed in the tree the *Course.Status* value is set to *Initiated*.

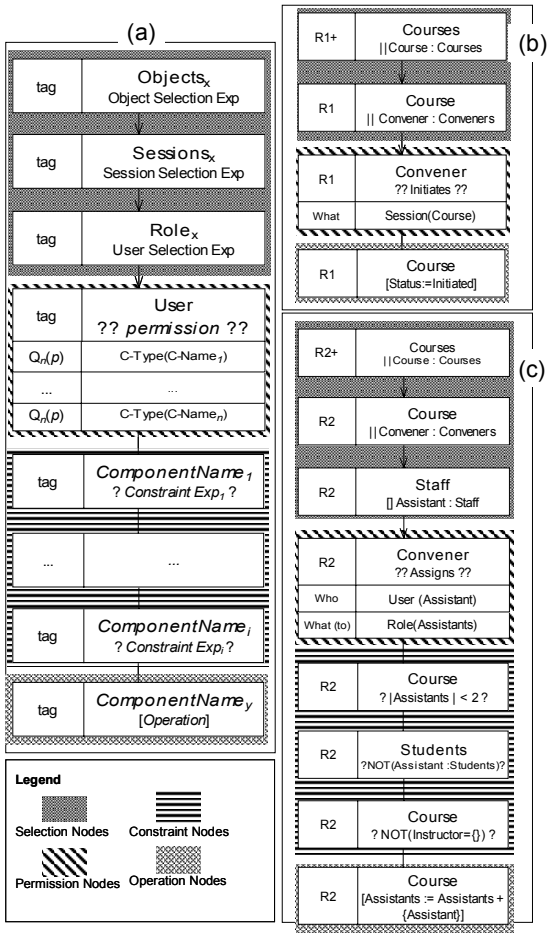

Figure 3: (a) The BT-RBAC model. (b) BT – RBAC specification for *R1*, (c) *R2*.

Similarly, the requirement *R2* from Table 1 is modeled as in Figure 3(c). The member of the *Conveners* role for a course is given the permission to assign members to the course's *Assistants* role. After selecting the course, conveners and staff members from their respective sets, the permission node is specified to permit the *Convener* to assign *User(Assistant)* to *Role(Course.Assistants)*.

When the event 'Convener Assigns' occurs, a number of constraints are checked before the 'add member to the Assistants role' operation is executed. The role *Course.Assistants* has a limit of two members on the cardinality of the set. Only if this condition holds the subsequent BT will be executed. The next constraint specifies that a member can only be assigned to the *Course.Assistants* role if he/she is not already a member of the Students role. Lastly, a member can be added to the Assistants role only if a member has already been assigned to the Instructors role. In the last node we add the chosen assistant to the set of course assistants.

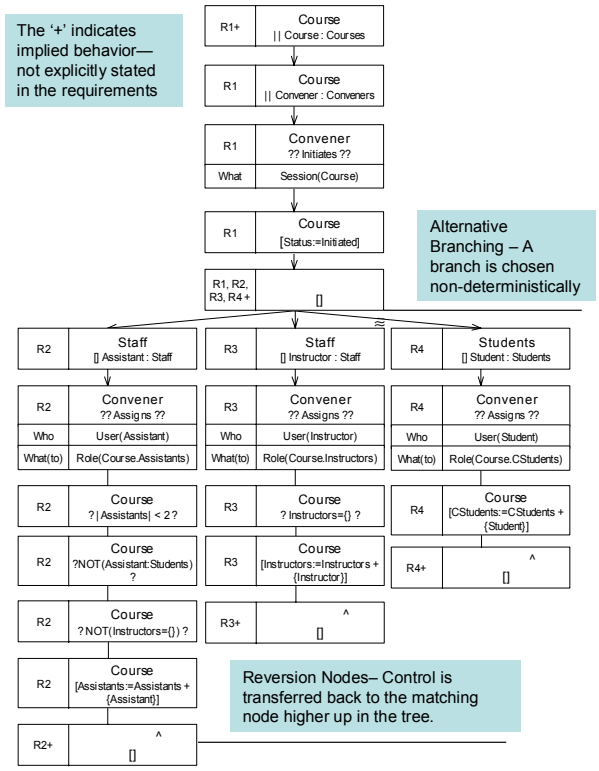

Figure 4: An IBT for *R1, R2, R3* and *R4*

After the individual requirements have been translated, they are integrated into an IBT, as outlined in Sect. 2.2. The integration of the requirements in Table 1 is shown in Fig 4. Functional requirements may also be freely integrated at this stage, since the BT notation can also express such requirements; for brevity, in this paper we focus on integration of security requirements.

4. Tool support

In this section we describe how the method is supported by simulation and model checking. The interface to these tools is managed by a BT editor, called *Integrare* [22]. The trial version of the tool is available at [http://www.behaviorengineering.org].

4.1. Validation via simulation

A BT may be executed via a simulation tool, BTSim that is based on an operational semantics for the BT notation [23]. Each "step" in the execution corresponds to the application of a rule of the operational semantics, and typically describes the effect a node has on the value of the components in the system. BTSim can generate a single trace of the system by randomly resolving nondeterminism (such as that introduced by concurrency), or can generate all traces of a (finite) system, subject to hardware constraints. The simulator can check safety properties of the system, and some basic progress properties. Using the simulator, it is possible to quickly check the typical operation of a system in a way that is straightforward for someone familiar with the BT notation.

The Classroom BT-RBAC model was initially simulated in BTSim to check that the basic behavior of the system conformed to the intention. The process uncovered typographical errors and places where the level of atomicity had to be increased, and pointed the way to some simplifications. This stage typically involved relatively small numbers of courses, students, etc. After this process, the model was checked against some of the security requirements outlined in the last section. The simulations in this phase involved increased numbers of courses, students, exams, etc. Properties 1, 2, 3, 4 and 7 were checked through several complex runs of the system, which resulted in modifications to the model to properly capture the intended behavior. Because the simulator has only a partial implementation of temporal logic operators, properties 5 and 6 were omitted from the simulations. The iterative simulation/validation and modeling process established reasonable confidence that the model satisfied many of the requirements under a number of typical executions, and that the model was therefore mature enough for an exhaustive formal verification of the security requirements.

4.2. Verification

Using the BT editor's facilities we automatically translate the BT model of the Classroom RBAC into the input notation of the SAL tool set [9]. This allows us to run checks for the satisfiability of temporal logic formulas (bounded and unbounded) as well as deadlock checks. In the past the translation of BTs into SAL code has been used successfully for supporting Failure Modes and Effect Analysis [10].

The translation to the SAL language was extended for the BT-RBAC model to handle the *forall* and *forone* constructs, and to deal with components of type set. Most set operations can be readily translated by making use of the pre-defined *set context* which provides the standard set operations for the SAL language (this context has been further developed by the work of Smith and Wildman [24] which describes a translation from the set-based language Z into the SAL language). To map the *forall* and *forone* construct we developed an unfolding algorithm which works as follows:

$$\| \, s : S \,.\, T(s). \rightsquigarrow T(s_1) \| \ldots \| T(s_n) \qquad (\textit{forall})$$

$$[] \, s : S \,.\, T(s). \rightsquigarrow T(s_1) [] \ldots [] \, T(s_n) \qquad (\textit{forone})$$

Assuming $S = \{s_1, \ldots, s_n\}$ is a static set, then n threads $T(s_i)$ are executed in parallel (*forall*), or one of *n* threads is non-deterministically chosen (*forone*). If *S* changes its content dynamically we consider the maximal size it can grow to instead of *n* (the maximal size of a set is given through its type, which is provided by the user in a textual representation in an accompanying file). The entrance to each thread $T(s_i)$ in this case will be additionally guarded with the condition that the element s_i is currently in the set *S*.

To render the model checking process feasible we have to limit the size of the set variables regarded by the system. For this purpose the user needs to input the maximal size of the sets. Since the sets *Students, Staff, Instructors* and *Assistants* for each course, etc., all have to be of the same type to be comparable, those sets will be of the same cardinality too. Due to hardware limitations we applied the model checker to small instances of the BT-RBAC model. After correcting one variable initialization error we were able to prove all the properties (1-7) of the model.

5. Related work

In recent years, a number of techniques and methods have been proposed to address the integration of security engineering with systems engineering from different perspectives. Not surprisingly, UML, being a widely used modeling language, has been the focus of

many researchers. Jürjens [8] has proposed UMLsec, which is an extension of UML with a formal semantics to support analysis of security properties in the modeled requirements. The graph-based formal semantics of SecureUML and View Policy Language (VPL) proposed by Basin et al [7] and Koch and Pauls [25], respectively, allows the designers to reason about the preservation of access control constraints in the final UML-based system models using the model driven approach. In recent work, support for interactive theorem proving has been provided for SecureUML [26]. However, in most of the UML-based approaches a more exhaustive and complex analysis is generally not supported. In comparison the BT-RBAC model provides model checking support for verification of critical properties.

The RBAC specification techniques that do support rigorous analysis do not always explicitly address their integration into functional requirements. In [27] and [17], the specification languages Alloy and SPIN, respectively, have been used to verify the security properties using a model-checker. However, the formalization of access control requirements can compromise the expressive power of more informal representation. In addition, highly specialized models make it difficult to integrate access control requirements with other functional requirements of the system [7] and often requirements traceability is lost. In comparison, the BT-RBAC model does not only provide automated support for validation and verification but also supports the integration of the access control requirements into the functional requirements.

Perhaps the closest work to our approach is the Secure Tropos development methodology, which provides a formal framework for integrating security requirements into the software development process [28]. Secure Tropos is an extension of a more generic agent-oriented development methodology. Since the underlying model is the same, the integration of the security requirements is possible from the early stages of the development process. We differentiate our work from Secure Tropos on the basis of a systematic translation process that allows us to develop a system out-of-its-requirements, the support for early defect detection, effective requirements traceability, and conflict resolution through the requirements integration process.

6. Conclusion

Although there is a greater emphasis on the need for better security in computer-based information systems, security requirements are still typically considered late in the development cycle and are often applied as an afterthought [3]. There are four major problems that are typically faced in effectively integrating RBAC models into the functional requirements of a system: (1) the informal models may not be formally verified for correctness; (2) the formally specified models may not be easy to understand and validate; (3) the often highly specialized formal models use different notation to the specification and design notation; and (4) the overall development process does not support early integration of security requirements into the design process, and requirements traceability is often lost. Based on a graphical notation with a formal semantics the BT-RBAC model provides support for automated validation and verification of access control policies. The integration of access control requirements is assisted through the use of a single notation throughout the development process. Other features of the model include a systematic translation process, early defect detection, requirements traceability and support for requirements transformation into the system design.

In this paper we have illustrated, through the use of a simple case study, how RBAC requirements can be modeled and validated. The validation results are used to correct the model before it is formally verified. We have not included functional requirements in the case study but the integration of simple access control requirements have been illustrated in [15]. Our future work includes optimizing the SAL code to improve the efficiency of the model checking process as well as providing tool support to formalize security properties of the model in LTL. Our long term objectives include extending the Behavior Engineering approach to other areas of security engineering like threat modeling and intrusion detection.

Acknowledgements

This work is partially funded by the Australian Research Council (ARC) under the ARC Centers of Excellence program. We thank our colleagues in the Building Dependability into Complex Computer-Based Systems project for helpful discussions.

7. References

[1] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Transactions on Information and System Security*, vol. 4, pp. 224-274, 2001.

[2] E. Ferrari, "Guest editorial: Special issue on access control models and technologies," *ACM Trans. Inf. Syst. Secur.*, vol. 8, pp. 349-350, 2005.

[3] P. T. Devanbu and S. G. Stubblebine, "Software Engineering for Security: A Roadmap," presented at International Conference on Software Engineering (ICSE 2000) - Future of SE Track, 2000.

[4] R. G. Dromey, "Formalizing the Transition from Requirements to Design," in *Mathematical Frameworks for Component Software - Models for Analysis and Synthesis*, *World Scientific Series on Component-Based Development*, J. He and Z. Liu, Eds., 2006, pp. 156-187, (Invited Chapter).

[5] R. L. Glass, "Is This a Revolutionary Idea, or Not?," *Communications of the ACM*, vol. 47, pp. 23-25, 2004.

[6] D. Powell, "Requirements Evaluation Using Behavior Trees - Findings from Industry," presented at 18th Australian Conference on Software Engineering, 2007.

[7] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From UML models to access control infrastructures," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, pp. 39-91, 2006.

[8] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development," presented at UML 2002 - The Unified Modeling Language : 5th International Conference, LNCS 2460, pp. 412-425, 2002.

[9] L. d. Moura, S. Owre, H. Rueß, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari, "SAL 2," presented at Computer Aided Verification: 16th International Conference (CAV 2004), LNCS 3114, pp. 496-500, 2004.

[10] L. Grunske, P. Lindsay, N. Yatapanage, and K. Winter, "An Automated Failure Mode and Effect Analysis based on High-Level Design Specification with Behavior Trees," presented at Fifth International Conference on Integrated Formal Methods (IFM2005), LNCS 3771, pp. 129-149, 2005.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, pp. 38-47, 1996.

[12] R. G. Dromey, "From Requirements to Design: Formalizing the Key Steps," presented at First International Conference on IEEE International Conference on Software Engineering and Formal Methods (SEFM 2003), IEEE Computer Society, pp. 2-11, 2003.

[13] R. G. Dromey and D. Powell, "Early requirements defects detection," *TickIT Journal*, vol. 4Q05, pp. 3-13, 2005.

[14] R. G. Dromey, "Climbing over the 'No silver bullet' brick wall," *IEEE Software*, 2006.

[15] S. Zafar and R. G. Dromey, "Integrating Safety and Security Requirements into Design of an Embedded System," presented at Asia-Pacific Software Engineering Conference, IEEE Computer Society, pp. 629-636, 2005.

[16] L. Grunske, K. Winter, and R. Colvin, "Timed Behavior Trees and their application to verifying real-time systems," presented at 18th Australian Conference on Software Engineering (ASWEC'07), 2007.

[17] T. Ahmed and A. R. Tripathi, "Static verification of security requirements in role based CSCW systems," presented at Eighth ACM Symposium on Access Control Models and Technologies, ACM Press, pp. 196-203, 2003.

[18] D. F. Ferraiolo, J. Cugini, and D. R. Kuhn, "Role-Based Access Control (RBAC): Features and Motivations," *Proceedings, 11th Annual Computer Security Applications Conference*, pp. 241-48, 1995.

[19] E. A. Emerson, "Temporal and modal logic," *In van Leeuwen, J., ed.: Handbook of Theoretical Computer Science*, vol. Volume~B, Elsevier Science Publishers, 1990.

[20] D. D. Clark and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *IEEE Symposium on Security and Privacy*, pp. 184-194, 1987.

[21] R. T. Simon and M. E. Zurko, "Separation of Duty in Role-based Environments," *Proceedings: 10th Computer Security Foundations Workshop*, pp. 183-194, 1997.

[22] L. Wen, R. Colvin, K. Lin, J. Seagrott, N. Yatapanage, and R. G. Dromey, "'Integrare', a Collaborative Environment of Behavior-Oriented Design," presented at 4th International Conference on Cooperative Design, Visualization and Engineering (CDVE2007), 2007.

[23] R. Colvin and I. J. Hayes, "A Semantics for Behavior Trees," ARC Centre for Complex Systems April 2007.

[24] G. Smith and L. Wildman, "Model checking Z specifications using SAL," presented at 4th International Conference of B and Z Users (ZB 2005), LNCS., Springer-Verlag 3455, pp. 85-103, 2005.

[25] M. Koch and K. Pauls, "An Access Control Language for Dynamic Systems – Model-Driven Development and Verification," presented at 12th International SDL Forum, Springer Berlin / Heidelberg 3530, pp. 16-31, 2005.

[26] A. D. Brucker, J. Doser, and B. Wolff, "A Model Transformation Semantics and Analysis Methodology for SecureUML," presented at MoDELS 2006, Model Driven Engineering Languages and System, LNCS 4199, pp. 306-320, 2006.

[27] G. Hughes and T. Bultan, "Automated Verification of Access Control Policies," Computer Science Department, University of California, Santa Barbara, CA 93106, USA, Tech-Report 2004-22, September 2004.

[28] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modeling Security Requirements Through Ownership, Permission and Delegation," presented at 13th IEEE International Conference on Requirements Engineering (RE'05), IEEE Computer Society, pp. 167-176, 2005.