2014-02

Modeling human-in-the-loop security analysis and decision-making process

Schumann, Michael A.; Drusinsky, Doron; Michael, James B.; Wijesekera, Duminda

IEEE

M.A. Schumann, D. Drusinsky, J.B. Michael, D. Wijesekera, "Modeling human-in-the-loop security analysis and decision-making processes," IEEE Transactions on Software Engineering, v.40, no.2, (February 2014), pp. 154-166.
https://hdl.handle.net/10945/56499

Modeling Human-in-the-Loop Security Analysis and Decision-Making Processes

Michael A. Schumann, *Member, IEEE*, Doron Drusinsky, James B. Michael, *Senior Member, IEEE*, and Duminda Wijesekera, *Senior Member, IEEE*

Abstract—This paper presents a novel application of computer-assisted formal methods for systematically specifying, documenting, statically and dynamically checking, and maintaining human-centered workflow processes. This approach provides for end-to-end verification and validation of process workflows, which is needed for process workflows that are intended for use in developing and maintaining high-integrity systems. We demonstrate the technical feasibility of our approach by applying it on the development of the US government's process workflow for implementing, certifying, and accrediting cross-domain computer security solutions. Our approach involves identifying human-in-the-loop decision points in the process activities and then modeling these via statechart assertions. We developed techniques to specify and enforce workflow hierarchies, which was a challenge due to the existence of concurrent activities within complex workflow processes. Some of the key advantages of our approach are: it results in development of a model that is executable, supporting both upfront and runtime checking of process-workflow requirements; aids comprehension and communication among stakeholders and process engineers; and provides for incorporating accountability and risk management into the engineering of process workflows.

Index Terms—Formal methods, information assurance, process modeling, software engineering, statechart assertions, verification and validation

1 Introduction

Processes are a fundamental component of most activities undertaken by humans. According to Webster's New Universal Unabridged Dictionary, a process is "a particular method of doing something, generally involving a number of steps or operations" [1]. In software engineering and information assurance, in particular, it is important that processes be understandable, documentable, and repeatable so as to ensure that the process outcomes are consistent and satisfy end-user requirements. The ability to formally represent and reason about human-based decision-making processes is a prerequisite for implementing these processes in information systems.

We developed a novel approach to process creation, documentation, checking, and maintenance. Our statechart assertion-based approach applies mathematical formalism to the engineering of processes. We focus on human-based processes, that is, processes that rely in large measure on human decision-making to advance the process flow; however, this modeling approach is sufficiently general for application to any type of process.

- *M.A. Schumann is with KEYW Corporation, 7740 Milestone Pkwy, Suite 400, Hanover, MD 21060. E-mail: mschumann@keywcorp.com.*
- *D. Drusinsky is with the Department of Computer Science, Naval Postgraduate School, Monterey, CA 93943. E-mail: ddrusins@nps.edu.*
- *J.B. Michael is with the Departments of Computer Science and Electrical & Computer Engineering, Naval Postgraduate School, 900 N Glebe Road, Arlington, VA 22203. E-mail: bmichael@nps.edu.*
- *D. Wijesekera is with the Department of Computer Science, George Mason University, Fairfax, VA 22030. E-mail: dwijesek@gmu.edu.*

Manuscript received 25 Jan. 2013; revised 8 Jan. 2014; accepted 20 Jan. 2014; date of publication 28 Jan. 2014; date of current version 4 Mar. 2014.
Recommended for acceptance by D. Drusinsky.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2014.2302433

Our approach utilizes statechart-based formal process modeling and the use of embedded statechart assertions to ensure the modeled process adheres to stated requirements, thus providing traceability[1] between the process requirements and the process implementation. Formal methods have been most frequently used in the software engineering of highly automated mission- and safety-critical systems. In contrast, our research demonstrates the use of formal methods to specify and reason about primarily human-based processes, such as the process used by the US government to implement, certify, and accredit cross-domain solutions (CDS). This process is titled the CDS Workflow and serves as a demonstrative exemplar of our modeling approach.

The intent of our approach is to impart a high degree of precision to our understanding of the process, as well as to provide an automated means of validating that the process exhibits only the behaviors we expect. In addition, our approach provides for runtime monitoring of the process. Runtime monitoring is useful for both validation which is about answering the question "Is our formal specification of the natural language description of the process correct?" as well as verification which is about answering the question "Have we correctly implemented the process?" Runtime monitoring is possible because one of the artifacts produced by our modeling approach is an executable representation of the process. In essence, runtime monitoring uses an executable version of the process and assertions about the process to evaluate input scenarios and classify them as good or bad [2].

1. We apply the IEEE 610.12-1990 definition of traceability stated as, "The degree to which each element in a software development product establishes its reason for existing; for example, the degree to which each element in a bubble chart references the requirement it satisfies."

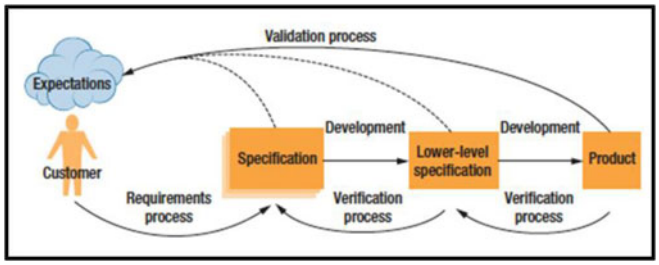

Fig. 1. A continuous V&V process from [4].

As Gabbar pointed out, the use of formal representation provides a systematic framework to construct and validate the syntax of the underlying system towards building standard representation approaches [3]. Michael et al. show us that (*vide* Fig. 1) we can translate customer requirements to formal specification then employ validation and verification (V&V) throughout the development process in order to ensure that the model satisfies stakeholder expectations [4].

We have analyzed the CDS Workflow process in terms of our modeling approach and used the approach to develop a formal process model. We do so using formal methods-based techniques and tools that support them such as those described in [2], [3], [5]. The resultant model provides the level of formality necessary for rigorously analyzing the exemplar process.

In order to understand a series of activities conducing to an end completely, especially when that end directly relates to the level of trust we place in high-assurance systems, we must be able to: rigorously articulate the process; consistently predict the process outcomes; and enforce requirements on the process and its individual steps.

In order to implement, certify, and accredit systems for operation at the highest levels of assurance, we need to be able to both understand and trust in the process through which we implement, certify, and accredit those systems. Ultimately, our work in defining a formal model of the CDS workflow process helps build the evidence necessary to certify and accredit high-assurance CDS systems. However, a process that is well-defined, validated, and documented is only enough to guarantee safety (i.e., ensure that nothing bad happens). The reification (i.e., mapping) from the formal model to the implementation is also needed to guarantee security properties. In other words, V&V of the process is a necessary but not sufficient condition for obtaining a trusted system. This insufficiency stems from the general non-computability results of other desired properties such as liveness and non-existence of covert channels in composable workflows.

The rest of this paper is organized as follows. Section 2 provides an assessment and discussion of related research. In Section 3 we introduce our systematic formal approach to process modeling. In Section 4 we present the results of applying the modeling approach to a real-world decision-making process. In Section 5 we present our conclusions and avenues for further refining and applying our approach.

2 Background Research

2.1 Introduction

There is a long history of research on the modeling of software and security processes, such as those used for the purposes of life-cycle development including but not limited to requirements elicitation, requirements specification, system development, testing, maintenance, and in particular for V&V, certification and accreditation (C&A). For example, the primary focus of the First International Conference on the Software Process held in Redondo Beach, California in 1991 was on high-level processes such as those by which software is developed as opposed to low-level processes such as those managed by an operating system (i.e., execution streams in the context of particular process states). Subsequent to the conference, the field of formal process modeling continued to evolve not only for software engineering processes but also for business processes. Researchers in both fields experimented with a wide variety of tools and techniques in the effort to find ways to formally specify, validate, and verify high-level software development and business processes.

Additionally, a significant body of research exists on the development of executable process models from formal specifications. However, a gap exists in the open literature with regard to conducting runtime verification of the executable process models. Our work addresses this need by providing a systematic process modeling approach that includes runtime verification of the executable process model via embedded statechart assertions.

2.2 Formal Methods in Process Modeling

Gruhn and Emmerich introduced a software process modeling language called FUNSOFT nets. These are essentially Petri nets with a formally defined semantics in terms of Predicate/Transition (Pr/T) nets and extended by multisets [6]. Gruhn describes software process modeling as: focusing on software process models that can be used for governing software processes with the intent of automatically detecting incongruities between a software development process and its associated model. Executable software process models, according to Gruhn, contribute to increased software development productivity and software quality. Gruhn extended the discussion of FUNSOFT nets to encompass social processes represented by software development teams [7]. He points out that the incorporation of human social interactions into software process models represents an area of research that is not well understood though he does suggest that the dialogue artifacts inherent in FUNSOFT nets may provide a means of describing these interactions in the context of software processes [6]. Gruhn concludes that the non-linear factors involved in human social interactions prevent the modeler from doing more than pointing out explicitly where human interaction and cooperation impacts software development.

Hibdon and Hartrum examine the development of an organizational process model based on object-oriented design concepts and formal software engineering methods. They apply a sequential design process that builds an informal Rumbaugh model, translates it to a Z based specification, and then translates the Z specification into

the Refine language and builds an executable model via the software refinery environment [8]. In our approach, the statechart-based model and the embedded assertions are designed with the same modeling techniques in the same formalized language.

An experimental application of process modeling technology at British Airways showed that there is value in using flexible modeling tools as the modeling process can reveal flaws and inconsistencies in the original process. If this results in a change to the original process, the model needs to be changeable to reflect the adjustments [9]. Our modeling approach addresses this issue by using a modeling tool that is flexible enough to apply rapid changes yet maintains consistency within the model.

Business process models have become increasingly complex, making it steadily more difficult to implement the models within an information system. Koehler et al. claim that a dichotomy exists between the tools and methods used to describe a business process and the tools and methods used to describe the information technology (IT) artifacts implementing the process. They make the case that process requirements should be made explicit and demonstrate the use of basic model checking techniques to verify a model's global properties of reachability and liveliness [10]. Here, the term "global properties" refers to those properties that apply to the entire model. While verification of these properties is useful in terms of a basic understanding of how the model specifies system behavior, it leaves the correctness of the enforcement mechanisms as outside of the verification domain. In other words, these properties are agnostic to the contents of the model. Our research addresses the need for contextual verification of a model's internal properties through the runtime application of embedded statechart assertions.

As pointed out by Gruhn and Lane, the building of business process models can benefit from well-established practices in software engineering [11]. Business process modeling notation (BPMN) is an emerging standard that allows business processes to be captured in a standardized format. The initial incarnations of BPMN lack formal semantics which leaves many of its features open to interpretation and hinders verification of processes described in BPMN. Ye et al. proposed a methodology for mapping a subset of BPMN elements to yet another workflow language (YAWL) specification language formal, set-notation based definitions [12]. Their work provided an initial step toward the type of formalization required for BPMN model verification. It also highlights the challenges of taking a model developed in a language lacking formal semantics and translating that model to a sufficiently formal language to allow verification. Wong and Gibbons demonstrated a technique for representing BPMN process models in CSP in order to provide a semantics for formal analysis and comparison of BPMN diagrams [13]. Our modeling approach also provides an alternative semantics and thereby provides the process engineer with the capability to both design and verify process models within the same formal specification language.

Grady offered a universal architecture description framework (UADF) and showed how the combination of UML and SysML can be applied to modern-day problem spaces to provide organized methods for identification of specialty engineering/quality, environmental requirements, product entities, and the map between models and product entities borrowed from traditional structured analysis (TSA) [14]. Grady points out the difficulty in connecting a design with the verification process through which we prove that a design satisfies its driving requirements. Our work in applying statechart-based assertions to the modeling of processes addresses this by modeling requirements in the same statechart-based notation as the modeled process and embedding those assertions within the modeled process to enable runtime enforcement.

The integration of human interactions into process modeling can be challenging due to the unpredictable nature of human behavior. It is important to find ways to formally specify human interactions within a process in order to facilitate process V&V. Zongyang et al. propose a formalization of human interactions within business processes through the introduction of the human processes and artifact (HP/A) model. This model applies rigorous, set notation-based definitions to human processes combined with statechart visual representations of the human interactions within a business process. However, the authors identify a gap in the verification of models that incorporate human interactions due to the ability of humans to make unpredictable choices [15]. Our work contributes to closing this gap in human-based model verification by integrating the representation of human choices within the process models designed through our modeling approach. As a result, requirements or constraints on human choices can be modeled and enforced at runtime using statechart-embedded assertions within an executable model, facilitating runtime verification of models that integrate the representation of human choices.

Hurtado et al. point out that the software process models can be sophisticated and large [16], [17]. As a means of reducing complexity, they demonstrate the analysis and visualization for software process assessment (AVISPA) tool for analyzing and identifying errors as an *a priori* way to measure the quality of the process prior to execution. Hurtado et al. point out that formal V&V techniques for measuring and testing discrepancies between a model and its execution can only be carried out on a process model that has been implemented, tailored, and enacted. Our process modeling approach addresses this concern by providing the process engineer with an iterative approach for use in process design that integrates V&V throughout and includes the development of an executable representation of the process model.

2.3 Safety

Bishop provides an introduction to the concepts of information leakage and safety in information systems. These terms are used rather than secure and insecure because safety refers to the abstract model and security refers to the actual implementation [18]. Safety as described by Bishop is critical for CDS. These systems must employ access controls that guarantee safety in order to prevent the inadvertent transfer or disclosure of sensitive or classified information. However we cannot analyze the process for developing a system in terms of its safety guarantees unless we precisely understand that process. Our statechart-based

approach to formal process modeling provides the level of precision necessary to analyze the process model in terms of its safety guarantees.

2.4 Statecharts

Harel introduced statecharts to address a well-recognized problem with regard to the difficulty of specifying and designing reactive systems [19]. A reactive system, in contrast to a transformational system, is characterized by being, to a large extent, event-driven and continuously reacting to external and internal stimuli, such as in automotive collision-avoidance systems or adaptive networks of wireless sensors. Harel's seminal work on visual specification has been applied in a wide variety of subsequent research on the application of statecharts and their successor, UML statecharts.

Dong and Shensheng demonstrate that statecharts can be used to model business workflows by modeling an international travel agency's process for handling customer travel requests. They show that it is possible to represent hierarchal levels of the workflow and transition between levels by leveraging the AND/OR decomposition of statecharts. Additionally, they suggest that the well-defined semantics of statecharts allow for the verification of statechart-based workflow models [19], [20].

Drusinsky applied UML statecharts to real-world specification and verification in [2], [21], [22]. Though the concepts and techniques introduced by Harel and Drusinsky focus on using statecharts for the specification and development of reactive systems, we show that these same techniques enable us to formally specify and reason about the largely human-based CDS Workflow process. Drusinsky uses a Java-based statechart notation (i.e., any Java statement can be written as a statechart action, any Java condition can be written as a statechart transition guard, and any Java method name can be written as a transition event) as a basis for describing reactive systems [2], [23]. In other words, this Java-based statechart notation is Turing-equivalent. The notion of Turing equivalence in our chosen notation is important as this equivalence relationship tells us that the language described by the notation computes precisely the same class of functions as Turing machines. Therefore, the deep body of research on the power of Turing machines applies to this Java-based statechart notation. For additional details, authors such as Sipser, Hopcroft, and Kelley provide a more complete discussion of the Turing machines and their range of computable functions [24], [25], [26].

Drusinsky and Shing extend UML statecharts to include K-statecharts [27]. K-statecharts support the use of knowledge logic formulae—a form of modal logic used for modeling and reasoning about multi-agent systems whose behavior depends on knowledge and belief statements made by the system agents. Their model provides inter-visibility amongst the agents. The ability of an agent to observe and act on the behavior (i.e., states) of another agent allows the development of formalized, executable models for multi-agent systems.

Crane and Dingel explore the syntactic and semantic differences between three different statechart formalisms: Classical, UML, and Rhapsody [28]. Their results indicate a lack of standardization between these formalisms. They show that due to subtle semantic and syntactic differences a model that is a well-formed statechart in all three of these formalisms may exhibit different behaviors in each of the separate formalisms; this is not a concern in our approach as we use a single formalism—UML statecharts.

2.5 Requirements

In the field of formal verification of systems or systems of systems, we ensure that the behavior of a subject system complies with its formal correctness specification. However, the formal specifications are typically based on natural language (NL) requirements specifications. Drusinsky points out that NL specifications can be ambiguous and we must be careful when writing formal specifications from NL specifications. This ambiguity creates the potential for differences in perception and interpretation of requirements by stakeholders. Thus we must ensure that the translation of NL requirements is as accurate and precise as possible. Several ongoing research efforts address this open problem. Bruegge and Dutoit articulated a UML-based model for requirements elicitation and analysis that demonstrates the capturing of customer requirements, typically in natural language and subsequent translation to formal or semi-formal notation. The transformation to a more formal notation ensures that system developers work from a common understanding of the requirements provided by system stakeholders [29]. Drusinsky showed us how to identify NL requirements of interest from UML analysis diagrams (e.g., activity and message sequence diagrams) [22].

It is important to validate formal requirements specifications to ensure they correctly represent the intended behavior. In the case of requirements specifications written as statechart assertions, Drusinsky et al. introduced a pattern-based methodology for validating them against their NL requirements; this is particularly useful when the assertions are grouped into libraries of reusable formal specification assertions. The underlying concept for this approach is that statechart assertions are often focused on a specific, coherent concern. This suggests a likelihood of ensuring they correctly represent the intended behaviors by testing them against a finite, representative set of validation scenarios. The pattern-based methodology uses representative groups of tests (i.e., patterns) such as obvious success, obvious failure, event repetition, and multiple time intervals to ensure that testing includes the type of scenarios often overlooked in the validation process [23], [30].

Our work addresses validation of requirements for human-based processes by facilitating a clear, visually appealing articulation of requirements in the same notation used to model a process under examination. When articulated in this manner, *post facto* analysis and modification of these requirements may be performed in a rigorous and well-structured manner. Our work addresses the requirements verification concern for software engineering-related human-based processes by ensuring process adherence to requirements articulated in statechart assertions and embedded within our statechart-based process models.

2.6 Research Gaps

In the preceding subsections we identified gaps in the wide body of research on process engineering and process V&V.

A significant body of research exists on the development of executable software development and business process models from formal specifications. However, a gap exists with regard to conducting runtime verification of the executable process models. Our work addresses this gap by providing a formal process modeling approach with runtime verification of the executable process model via embedded statechart assertions. We achieve this by treating human-based processes as reactive systems.

From the perspective of applying formal methods in real-world projects, it is not realistic to expect engineers to construct a model in a language lacking formal semantics and then translate that model to a sufficiently formal language to allow for V&V. Our approach addresses this problem by using a single formal specification language for construction, validation, and verification of the model.

3 Process Modeling Approach

3.1 Assessment of Formal Methods Tools and Techniques

3.1.1 Desirable Attributes of Formal Methods to Support the Modeling Approach

The prior work reported on in Section 2 revealed knowledge gaps in the field of applying formal methods to high-level processes. We developed a set of desirable attributes for a formal specification language and associated tools to support the modeling of high-level, human-based processes and address the identified knowledge gaps.

The language should have a well-defined syntax and semantics in order to provide the level of formality necessary to unambiguously model and facilitate V&V of the process. The visual representation of the language should be sufficiently understandable as to provide a good communication medium between users, stakeholders, and process engineers. The language needs to provide mechanisms or artifacts that allow the formal specification of human-in-the-loop decision-making. The language needs to be able to represent attributes such as hierarchy and concurrency that are often found in complex human-based processes. The formal specification of the model and the artifacts necessary for verification of the model should all be expressible in the same language. Table 1 compares several languages using the aforementioned desirable attributes.

The language's associated tools should provide a mechanism for building a visual representation of the models using the chosen language. The modeling tools should provide the flexibility to make adjustments to the process model as necessary. They should also be able to generate an executable representation of the model. The associated tools should provide the ability to conduct runtime verification of the modeled process and an automated means of verifying an executing model's adherence to specified properties or requirements. This would require a means of monitoring the process model in execution and enforcing desirable runtime properties of the executing process model, such as ensuring adherence to temporal constraints.

We hypothesized and demonstrated via case studies that a formal language and associated tools that have, at a minimum, the listed attributes will enable the formal specification, maintenance, validation, and runtime verification of a model that accurately represents a partially automated, human-based, C&A process and provide a viable communication medium for discussing the process among users, stakeholders, and process engineers [31].

TABLE 1
Some Desirable Attributes of a Formal Language for Process Modeling Approach

Description of Attributes	Language: CSP	Petri Nets	Z	UML Statecharts
Well-defined syntax/semantics	☑	☑	☑	☑
Easily understandable (visual)	☐	☑	☐	☑
Specify human decision-making	☑	☑	☐	☑
Represent hierarchy	☐	☐	☑	☑
Represent concurrent activities	☑	☑	☑	☑
Model/verify in same language	☑	☐	☑	☑

3.1.2 Assessment of Formal Methods for Desirable Attributes

We assessed several formal languages and their associated tools, such as communicating sequential processes (CSP) [32], Petri nets [6], the Z formal language, and UML statecharts [2] to determine whether they possessed the attributes listed in Section 3.1.1. These languages are compared in Table 1 on the basis of how closely each one matches the set of desirable attributes.

Ryan and Schneider used CSP as a modeling mechanism for a variety of security protocols [33]. Wong and Gibbons demonstrated a technique for representing BPMN process models in CSP in order to provide a semantics for formal analysis and comparison of BPMN diagrams [13]. As discussed in Section 2 of this paper, CSP is based on a relatively complex mathematical notation. Many researchers in the field of process modeling have discussed the importance of reducing the complexity and increasing the ease of understanding formal process models in order to be better able to communicate with process users, stakeholders, and process engineers [9], [10], [15], [16], [17], [34].

There have been attempts to apply Petri nets to high-level processes as a means of formal specification. The works of Emmerich and Gruhn [6], Gruhn [7], Van Der Aalst and Van Hee [35], and van Dongen et al. [36]

demonstrate a variety of methods of applying Petri nets as a tool for process modeling. The graphical nature of Petri nets makes them an excellent tool for communicating about a process under examination. However, researchers have pointed out that Petri nets do not scale well for the visual representation of large, complex processes as the basic Petri net formalism lacks artifacts for representation of hierarchy. Clempner proposed an extension to represent hierarchy in a subclass of Petri nets known as decision process Petri Nets (DPPNS) though his work is still in a formative stage [37].

Hibdon and Hartrum built an executable model of a US Air Force component known as a wing. In this context "wing" refers to an organizational unit of the US Air Force vice a physical component of an airplane. Their modeling process required creation of an informal Rumbaugh object model, translation to the formal language Z, and subsequent translation into Refine constructs for execution [8], [38]. The final product of their multi-step approach is an executable model. However the Z language and the Refine construct are both complex, non-visual representational formalisms.

UML statecharts are a visual formalism that has been used for specification of systems, architectures, and processes. Researchers have demonstrated that UML statecharts have well-defined semantics with artifacts expressive enough to capture elements of human-in-the-loop decision-making (*vide* Section 2). They have been shown to be an effective visual medium for communicating about processes [20]. Specifications written as statechart assertions and embedded within a statechart-based process model enable runtime verification of the model. UML statechart assertions are a class of statecharts and as such written in the same language.

UML statecharts possess the desired attributes for a formal specification language that we outlined in Section 3.1.1. This formal language was the best fit in terms of its potential for use in addressing the research gaps identified in Section 2.

After selecting a language that satisfies the stated desired attributes, we needed to determine whether any of the currently available tools for working with UML statecharts would satisfy the requirements stated in Section 3.1.1. Both research protoype and commercial tools exist for the design and manipulation of statechart-based models. We looked into several of these tools such as VisualSTATE, Yakindu, and StateRover [2], [39], [40].

VisualSTATE is a standalone statechart-based modeling tool that also provides a point-and-click interface for easy development and editing of models. This software has a built-in module for code-generation to automatically create an executable representation of the model in C++. The software automatically performs syntactic verification to ensure model compliance with the underlying language rules. The verification module includes the functionality for static analysis of the model to ensure compliance with both pre--defined and custom properties. VisualSTATE has a dynamic analysis module that can provide an animated view of how specific events affect a model. Events are fed into the simulation via an interface with the ability to replay sequences of logged events. However, VisualSTATE does not provide for enforcing requirements in conjunction with runtime execution monitoring.

Yakindu is a statechart-based modeling tool that operates as a plug-in for the Eclipse integrated development environment (IDE). Yakindu supports visual specification and automated generation of an executable model. The tool also has a point-and-click interface for building and dynamically adjusting models. The Yakindu plug-in can interface with an external code-generator module capable of mapping a statechart model to C or Java source code. However, the code generator module is experimental and must be installed separately from the Yakindu plug-in. The plug-in applies automatic syntactic verification rules to each statechart model and reports discrepancies to the model developer via visual and textual cueing. Yakindu provides a simulation function that executes the generated code but it does not support runtime verification.

StateRover is a UML statechart-based modeling plug-in for the Eclipse IDE. This tool includes a built-in code generation module that automatically maps a statechart model to C, C++, or Java source code. Model design is accomplished through a point-and-click interface that makes it easy to build and dynamically adjust or reuse models. The tool includes an automated syntactic and semantic validation module to ensure model compliance with the underlying statechart syntax rules and semantics. In addition, code generation will not run unless the model is able to successfully pass the syntactic and semantic validation with no errors. StateRover provides the functionality for runtime verification of a statechart model through the application of embedded statechart assertions enforced within the model during execution. StateRover provides an integrated, white-box test generator that builds test cases for use in automated testing. The generated test cases are used within the JUnit test framework to provide runtime execution monitoring of the model as it enacts the generated test cases. Embedded statechart assertions serve to enforce runtime properties or constraints placed on the model (e.g., temporal constraints).

Of the statechart modeling tools surveyed, StateRover possesses the desired attributes for a tool (*vide* Section 3.1.1) designed to enable modeling in our chosen formal language. This tool was a best-fit in terms of its potential for use in addressing the research gaps (*vide* Section 2).

3.2 Procedure

The diagram shown in Fig. 2 outlines our process modeling approach. Solid lines represent the process flows. Dashed lines represent ongoing communication with process-stakeholders to ensure a modeled process aligns with and achieves their desired outcomes as originally specified in stakeholder requirements. This ongoing feedback loop is one component of model validation.

Our approach provides the level of formalism necessary to rigorously specify a partially automated, human-based, C&A process and conduct runtime verification on that process to ensure the process behaves exactly as it was designed. This allows the process engineers and stakeholders to ensure that the process flows exactly as it is intended. Formalization has the potential to improve the final product of a process for developing, implementing, and certifying and accrediting cross-domain solutions designed to facilitate and guard the flow of information between various security domains.

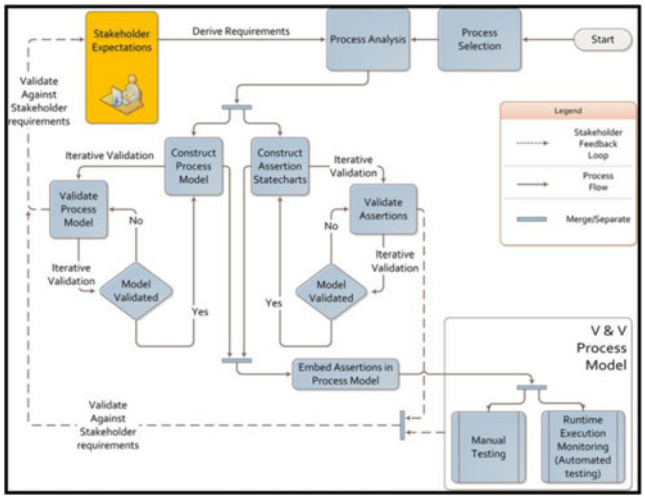

Fig. 2. Overview of statechart-based process modeling approach.

Formalizing the process can help ensure that it provides a product that is well-defined, well-developed, and consistent in its execution.

3.2.1 Iterative Design

Our modeling approach allows the process engineer to iteratively create a process model in conjunction with creation of the process itself. Once the process is established, the process engineer can use this approach to adjust the model throughout the lifecycle of the underlying process.

3.2.2 Terminology

We define the following terms in the context of human-based processes:

Thread. An activity that is orthogonal to other activities within the same process.

Process step. An atomic action within a process.

Sub-process. A set of two or more steps grouped together based on their degree of cohesion and coherence.

Transition. Movement from a step, sub-process, or statechart assertion to another.

Decision point. Places where a decision must be made (e.g., yes or no, approve or disapprove) and that decision's outcome falls within an expected range of values. The decision is modeled as a conditional- or value-based transition between components within the model.

Timing. Quantifiable time constraints or restraints, such as the time at which something must occur or the time within which something must occur.

Complexity. A metric expressed as a summation of joins and splits (AND, OR, or XOR) in a process.

Layering. Representing nested levels within a process model.

Scenario. A combined collection of desired properties, timings, human decisions, and/or conditions applied *vis-à-vis* a process model in order to exercise or stimulate various aspects or behaviors of the model.

3.3 Modeling Approach

In this section we provide the details of each step of our process modeling approach shown in Fig. 2.

3.3.1 Process Selection

Process selection is a non-trivial matter. Informal processes or well-defined processes whose outcomes are not mission- or safety-critical may not require the level of formality afforded by our approach. The application of our statechart-based modeling approach requires a time investment to analyze the process and apply formal methods. However, it is a worthwhile investment for the types of processes that we are interested in formally specifying and reasoning about. The modeled process will be easier to understand and communicate about, while the resulting process model will be amenable to testing, debugging, and maintenance.

3.3.2 Process Analysis

Prior to constructing the process model in StateRover, it is helpful to analyze the process. During this phase of our modeling approach we identify components of the process that can be specified as artifacts in a statechart-based formal model. This is also where we identify specific requirements that we model as assertion statecharts using formal specifications. Process analysis facilitates the development of a robust, granular, and high-fidelity model via the thorough a priori inspection of the process during model development. Similarly, process analysis provides the process engineer an opportunity to identify many if not all of the process components and timing considerations.

Process formalization requires thorough analysis of the chosen process as a key component of developing the formal model in StateRover. Through analysis, we develop an understanding of the process under examination and begin to formulate a plan for contextualizing individual components *vis-à-vis* our modeling approach with its associated views and terminology. We must understand threads, transitions, decision points, process requirements, timing, complexity, layering, and important steps in the process.

The process engineer uses a combination of available sources such as informal drawings, interviews with stakeholders, mission statements, modeling diagrams (e.g., UML activity diagrams, YAWL workflow charts), or basic flowcharts. The focus of this phase of our modeling approach is to develop as complete an understanding of the process as possible. One hurdle in process analysis is that the stakeholders' understanding and documentation of the process could range from the subject matter expert's domain-specific knowledge held by one or a few individuals to more flowcharts or other types of models based on notations like BPMN.

During this phase of the modeling approach the process engineer also gathers requirements from stakeholders in the process. These requirements will provide the source material for developing embedded statechart assertions—a key element to enable runtime execution monitoring of the process model.

3.3.3 Construct Process Model

It is during this step of the modeling approach that the process engineer builds the statechart-based process model. Leveraging the products of the "Process Analysis" step the process engineer visually specifies the process using the

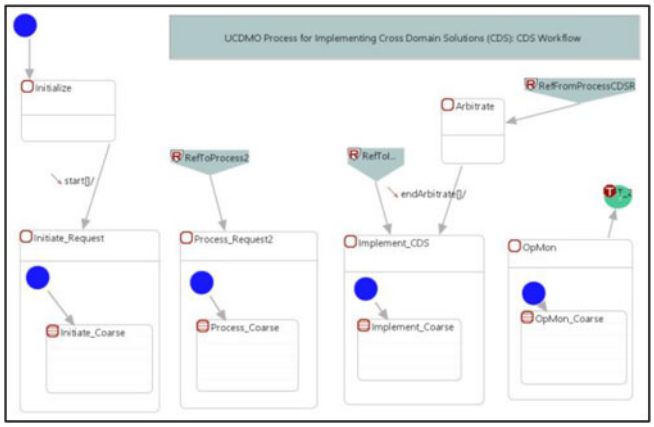

Fig. 3. Top-level statechart model of CDS workflow process.

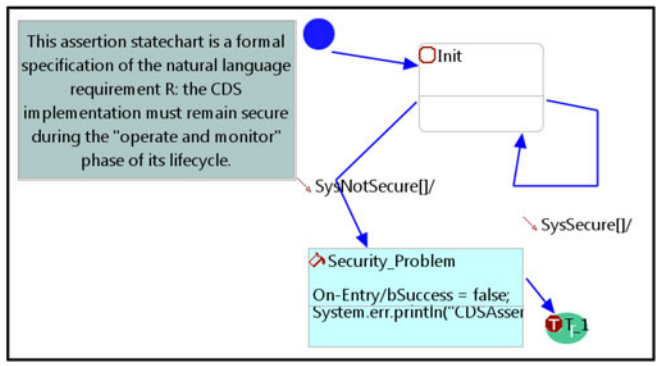

Fig. 4. Example of a statechart assertion.

formal language and tools chosen for their adherence to the desirable attributes listed in Section 3.1.1.

We previously demonstrated the novel use of UML statecharts as a medium and the StateRover modeling tool as a mechanism for formally modeling the cross domain implementation process (CDIP), a partially automated, human-based, C&A process [41]. In this paper we demonstrate the use of UML statecharts as a fundamental component of the process modeling approach shown in Fig. 2.

Because our chosen modeling tool generates an executable model in Java we are able to add Java code to just about any component of the model such as states, transitions, or flowchart boxes. This helps ensure that embedded assertions are enforced at runtime. UML statecharts provide a vehicle for the articulation of, formalization of, and communication about a process model such as the one shown in Fig. 3, a process model for our demonstrative exemplar, the CDS workflow process. In addition, we are able to take full advantage of automated statechart-handling capabilities built into StateRover such as hierarchy, concurrency, non-determinism, syntactic validation, workflow modeling, automated testing, and runtime monitoring.

During the design process, the process engineer is able to use the immediate feedback from StateRover's rule-checking mechanisms to identify possible errors within the process model. This is an enabler for applying a systematic and iterative approach to detecting and removing errors and improving upon the process model.

3.3.4 *Construct Statechart Assertions*

In this step, the process engineer transforms the requirements developed during "process analysis" into statechart assertions. The UML statechart specification of each requirement is in the same statechart-based language as the rest of the model. This provides a precise way of stating requirements that directly takes advantage of the formalisms used to develop the process model. During the execution phase of our approach, embedded statechart assertions are employed as enforceable runtime specifications.

Embedded statechart assertions are a key ingredient of our approach: They facilitate runtime execution monitoring as well as enforcement of desirable properties or requirements placed on the process (e.g., submission timeline for a request necessary for process progression).

Statechart assertions have two fundamental differences from the statecharts used throughout the rest of our modeling process: (i) they have a built-in mechanism for indicating Boolean success or failure (true/false), which makes them suitable for formal specification and (ii) they can be nondeterministic if desired [2]. Fig. 4 shows a statechart assertion specifying the NL requirement "R1: The CDS implementation must remain secure during the 'operate and monitor' phase of its lifecycle."

Sindre and Opdahl postulate that a visually appealing approach may actually be more successful than a textual approach when capturing requirements. This is because simple and intuitive diagrams provide a better overview of the functionality of a system and make it easier to see each stakeholder's interest in the system which in turn makes it easier to communicate about the captured requirements [42]. The combination of UML statecharts and embedded statechart assertions provides us with a visually appealing formal process modeling approach wherein the model and the assertions that enforce properties of the model are written in the same language, in this case UML statecharts. This addresses one of the challenges seen in previous formal process modeling research, that being the integration of a separate formal specification language in order to add formalisms to the process modeling approach [6].

According to the Oxford English Dictionary, an *oracle* is "an opinion or declaration regarded as authoritative and infallible" [43]. Since both our process model and the statechart assertions that represent requirements on the model are derived from NL descriptions, it cannot be assumed that one is more of an oracle than the other. However, the properties upon which we base statechart assertions are typically small enough that they do not require more than five to ten validation tests. This suggests that we can use a relatively small number of tests to build a body of evidence for using the assertions as an oracle for testing the behaviors of a process model. We adopted the pattern-based methodology of Drusinsky et al. because it helps ensure that we cover often overlooked testing areas when writing validation tests for our assertions [23], [30]. They describe scenario test patterns such as *obvious success*, *obvious failure*, *full scenario success*, and *full scenario failure*. Fig. 5 shows a test scenario to ensure that an assertion succeeds when it is supposed to for a simple set of

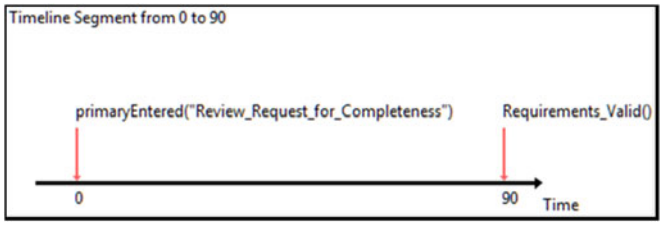

Fig. 5. Timeline diagram for "obvious success" assertion test scenario.

conditions (i.e., obvious success test pattern). Once the assertion has entered state *Timer* it must see a *Requirements_Valid()* event within 100 time units or the assertion fails. The event *Requirements_ Valid()* occurs at time 90 and no further events or transitions occur so we expect this assertion to succeed during testing.

3.3.5 Embed Assertions in Process Model

During this step of the modeling approach, representative artifacts for each statechart assertion are embedded in the process model.

We previously examined testing of statechart-based formal process models and showed that embedded assertions could be applied to formal models of human-based processes as a means of enforcing requirements [44]. When testing the model, failures to adhere to the requirements of the assertion are recorded and reported by the testing module. This ensures that the model behaves as expected under a wide variety of conditions while the executable version of the model is running. This technique allows us to use embedded assertions as an enforcement tool for process requirements because the embedded assertions are located within the model, which provides unique access to the process model's events, variables, and timing structures as it executes. This positioning is the enabler for embedded statechart assertions to act in an enforcement role. The *SecAssertion* box of Fig. 6 is an example of the method by which a statechart assertion is embedded within a process model. The *SecAssertion* box acts as a placeholder and insertion point for the statechart assertion of Fig. 4. An added advantage of embedded statechart assertions is that they are naturally scoped by the context of their substatechart [2]. Hence they are only active when their substatechart is entered and they cease to be active when the process transitions out of the containing substatechart. This property of embedded statechart assertions lends itself to hierarchy and scalability in the process modeling approach. This property also allows the process engineer to better deal with process complexity by providing the ability to enforce process requirements at runtime at a variety of levels and with varying scope. This also facilitates development, testing, debugging, and maintenance as the process engineer can ascertain the scope of each embedded statechart assertion.

3.3.6 V&V of the Process Model

During this step of the modeling approach, we leverage the components and capabilities of our chosen formal modeling tool to validate and verify statechart-based process models.

Our approach uses three types of validation: (i) automated syntactic validation via algorithms built into our chosen modeling tool; (ii) validation of statechart assertions to ensure they accurately represent stakeholder requirements; and (iii) validation against stakeholder expectations which requires process engineers to compare the model to requirements derived during the Process Analysis step of our approach and to maintain a review and feedback loop with process stakeholders. The intent of validation is to ensure the process model remains synchronized with stakeholder expectations throughout the design process; in Fig. 2 we use dashed lines to delineate the feedback loop.

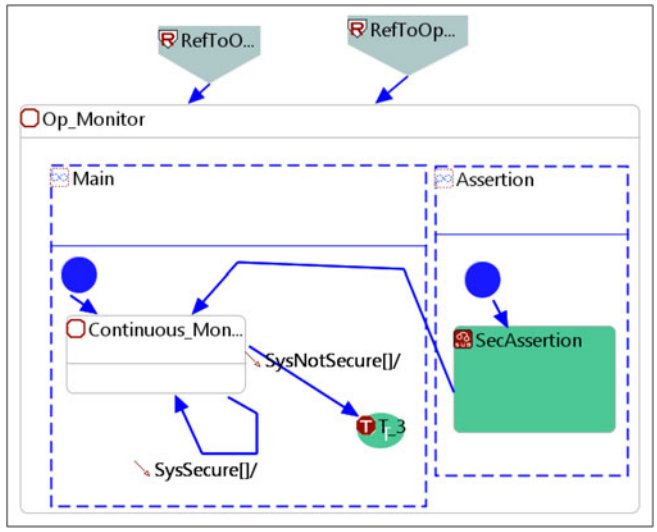

Fig. 6. Statechart assertion scoped by substatechart Op_Monitor.

We employ two types of verification in the modeling approach: manual testing and runtime execution monitoring. Each of these uses test cases which are an important construct for verifying that the executable version of a process model operates as intended. For the type of partially automated, human-based, C&A processes that we are interested in modeling, test cases equate to sequences of real-world process-related events, conditions, timing, and human decisions. Our objective here is to examine the response or flow of a process model in a simulated test environment.

The process engineer uses manually generated test cases for multiple purposes. During the Construct Process Model and Construct Statechart Assertions steps, the process engineer manually develops tests to iteratively ensure components of the model behave as expected as well as that the model as a whole accurately reflects the process being modeled and that the process produces expected results for specific scenarios. The engineer also constructs tests to examine one or more portions of the process during development, testing, debugging, and maintenance.

During the manual testing phase, the process engineer works at a micro scale, setting variables and sending events to an executing process model via handcrafted test cases. In contrast, during automated testing the tester works at the macro scale, adjusting test parameters such as number of tests, test length, number of permissible loops or choosing between stochastic and deterministic testing algorithms while a white-box test generator (WTG) automatically adjusts the micro-scale elements at runtime. The process engineer is able to use the information from testing to better

TABLE 2
Runtime Execution Monitoring Data Collection

Description of Attributes	Test Run # 1	2	3	4
Number of tests per run	5	25	50	100
Failed assertions	☑	☑	☑	☑
All states visited	☐	☐	☐	☑
Time to complete test run	77.5s	398.2s	853.9s	1523.6s

understand, test, debug, maintain, and communicate about a process model.

The executable version of a process model provides a vehicle for runtime monitoring. The medium within which this vehicle operates is the JUnit test framework, permitting the process engineer to apply at runtime automatically or manually generated test cases against the executable representation of the model. Automatically generated test cases facilitate exploration of all possible execution paths available to the executable model and the effect of a large number of input sequences on the process model.

We track the results of testing through tabulating and comparing such things as the number of tests per test run, whether assertions failed in a test run, whether all states were visited during testing, and the time to complete each test run, as shown in Table 2, in support of continuously improving the process model and the process itself.

4 Case Studies

In [31] we apply the process modeling approach (*vide* Fig. 2) in two case studies that demonstrate the utility of our approach. The first case study examines the cross domain implementation process (CDIP) and the second examines the CDS Workflow. Early on in our research, the CDIP was being developed by the US government's Unified Cross Domain Management Office (UCDMO) as the next-generation process for requesting, developing, implementing, and certifying and accrediting a cross-domain solution. We applied our process modeling approach to the CDIP. This effort helped refine our approach.

During the course of our research, the UCDMO transitioned from the CDIP to the CDS Workflow process. These processes are related in that the CDS Workflow process is an evolved form of the CDIP. When the UCDMO transitioned from the CDIP to the CDS Workflow as the process responsible for governing the request, development, implementation and C&A of cross-domain solutions, we began to model the CDS Workflow process too. This effort provided us with a number of benefits: application of our statechart-based modeling approach to two separate processes; fully exercising the runtime-monitoring capabilities of the modeling approach; and validating the ability to apply process requirements as embedded assertions and enforce those assertions on an executing model of the process. We provide below additional details on the CDS Workflow case study as it represents the more complex of the two case studies.

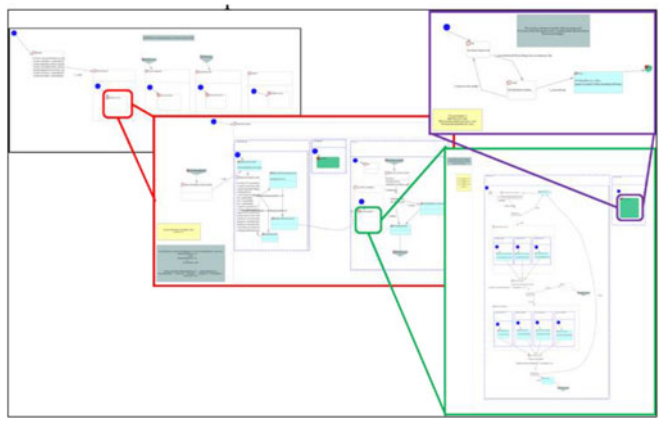

Fig. 7. Decomposing a complex hierarchical process model.

4.1 Cross Domain Solutions Workflow

4.1.1 Process Selection

The CDS Workflow replaced the UCDMO's CDIP with the former representing a process-based initiative to federate the request, reuse, development, implementation, and C&A of cross domain solutions. The CDS Workflow is a more complex process than its predecessor with five major process blocks and four levels of hierarchy as illustrated in Fig. 7 (*vide* [31, p. 44] for a full-size version of image). The diagrams shown in Fig. 7 demonstrate the successive decomposition of the four levels of hierarchy. When viewed together, we see that the ability to deal with complexity and layering in this way makes it possible to formally model hierarchical processes.

4.1.2 Process Analysis

In an effort to better document the CDS Workflow process, the UCDMO captured elements of it as UML use case and activity diagrams developed in Rational Rose Modeler.[2] The use case and activity diagrams were a starting point for analyzing and understanding the process, which facilitated the analysis phase of our modeling approach.

While most of the CDS Workflow subprocesses were documented individually through activity diagrams, one of the challenges was to develop an understanding of how the different activity diagrams related to each other in terms of processes, subprocesses, and sequencing in order to provide us with the necessary information to build a model reflective of the process. One subprocess, "Operate and Monitor," was not documented in an activity diagram. It was shown only as a single point on a use case

2. Rational Rose is a commercial UML modeling tool developed by IBM. Additional information is available at http://www-01.ibm.com/software/awdtools/developer/rose/modeler/.

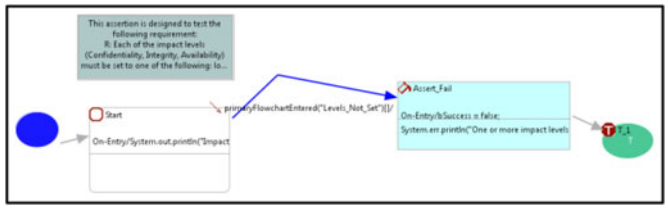

Fig. 8. Statechart Assertion for Requirement R1.

diagram. This prompted us to hold further discussions with the UCDMO representatives in order to determine the flow, elements, and desired behaviors of this subprocess. This demonstrates one of the challenges of applying formal methods tools and technique to processes operating in real-world environments.

As prescribed by our modeling approach, we also determined *threads*, *decision points*, and *layers* during this phase.

4.1.3 Construct Process Model

In this phase of the modeling approach, we bring together the results of process selection and analysis to articulate the formal process model of the CDS Workflow.

Throughout model development and with the addition of each new step we perform two actions designed to ensure the model adheres to the underlying rules for semantic correctness: (i) use the "Diagram/Validate" menu option to initiate StateRover's validation routine and reveal detected errors and (ii) initiate the code generation process. Diagram validation and code generation provide an end-to-end semantic check of the model and identify errors.

For this process, our analysis indicated that the top-level of the model would be relatively simple in terms of the number of states and transitions. We initially constructed the model shown in Fig. 3. This portion of the model ended up being very close to the final top-level view. Next, we constructed in turn each of the subprocess detail views for the *Initiate_Coarse*, *Process_Coarse*, *Implement_Coarse*, and *OpMon_Coarse* states of Fig. 3, each time performing the validation and code generation routines to ensure the model is semantically and syntactically correct.

4.1.4 Construct Statechart Assertions

In this phase of the case study we showed three embedded assertions statecharts. Each of these is designed to model and enforce a different NL requirement. Fig. 8 depicts a formal specification of requirement R1: *Each of the impact levels (confidentiality, integrity, availability) must be set to one of the following: low, moderate, high.*

4.1.5 Embed Assertions in Process Model

The next step in our process modeling approach is to embed the statechart assertions into the process model. Here, the process engineer places embedded statechart assertions within the model based on the desired scope of the assertion. The substatechart artifact of our chosen modeling tool ensures that an assertion is only applicable to activity modeled within its containing state.

4.1.6 V&V Process Model

The process engineer validates the finalized model by presenting it to the stakeholders of the process to ensure the model meets the stakeholders' expectations. For the purposes of demonstrating the feasibility of this component of our modeling approach, we worked directly with the UCDMO to ensure the completed process model met their expectations.

We informally presented diagrams of the formal model and discussed the translation process from UCDMO supplied process documentation to the process model. The resulting dialogue provided validation that our design met the UCDMO's intent. We used this feedback loop to ensure that the "Operate and Monitor" component of the model, mentioned above, met UCDMO expectations. Due to the lack of documentation available for analysis of "Operate and Monitor" it was particularly helpful to have direct input from the process stakeholders to improve the process model.

The modeling approach calls for model verification through both manual and automated testing. We used manually generated test cases to demonstrate how statechart assertions embedded in the CDS Workflow enforce requirements on the process model. When we write manual test cases, we are able to test process flow and adherence to process requirements written as embedded assertions by specifying events and setting variables to move through the executing process in a specific way. This ensures that the model behaves as expected for each test scenario.

We performed automated testing of the CDS Workflow by using runtime execution monitoring which explores the effect of numerous input sequences on the process model during automated testing and verifying that the model behaves as expected across the range of inputs. During model construction we insert code to print variable values and status messages at runtime. These messages are delivered throughout execution of each test. This helps us ensure that the model behaves precisely as specified since we are able to compare variable values for a test run to the testing results based on those values. Depending on our testing goals and the outcome of test runs, we adjust test parameters such as number of tests, test length, number of permissible loops in order to ensure all elements that we wish to examine have been tested.

5 Validation

In the case studies, we use statechart assertions that reflect the typical hard-to-model aspects of requirements on human-based processes and we demonstrate the modeling and V&V of several requirements for each modeled process. For the purposes of demonstrating the points in this research, formalizing all possible requirements on each process would not demonstrate anything additional. We recommend this as future work. A full-scale implementation of all process requirements would include approximately 120 embedded statechart assertions for each modeled process.

We believe the CDIP and CDS Workflow process are particularly well suited for use in experimenting with our approach, given that these processes involve human decision making, temporal constraints and restraints, nested subprocesses, workflow elements, and state changes.

As we developed the CDIP and CDS Workflow models there were some key lessons learned that will facilitate the improvement of further research into the modeling of partially automated human-based security-related processes.

The code generation capability turned out to be a key means of ensuring end-to-end syntactic and semantic consistency of our model. Note that during model development we do not focus on code generation from the standpoint of creating usable code, as would be the case when developing software-based systems. Instead, we are interested in the rigorous syntax and semantics checking that is an inherent part of StateRover's code generation module. If code generation is unable to complete successfully this equates to a syntactic or semantic inconsistency with the model and drives the process engineer to investigate and correct the source of the error before continuing with model development.

During the "Process Analysis" phase of our modeling approach, the process engineer develops an understanding of the process under examination based on the materials provided by the stakeholders of the process. This material may be abundant or scarce and could include things such as interviews with users or stakeholders, focus groups, informal diagrams, or semi-formal diagrams. We modeled two processes during the course of this research. The source material differed significantly between the two processes and we found in the case of the CDS Workflow that having the UCDMO provided activity diagrams and use cases facilitated our understanding of the model.

If provided a similar set of activity diagrams for any process, it would facilitate an expeditious analysis of the process. However, the notion of "garbage in, garbage out" (GIGO) applies here. In other words, if it is the case that either the process diagrams or documentation is inaccurate or incomplete, we believe that analysis could end up taking longer or result in unfounded conclusions due to the original diagrams leading the analysis down one or more false paths. Process diagrams are not a requirement for analysis but instead a facilitator. In the absence of diagrams or other forms of documentation, the process engineer engaged in formalizing a process is likely to employ a variety of methods to develop a full understanding of the process such as observation of the process in action and interviews with stakeholders. Based on our experience with the two processes modeled in this work, we suggest that inaccurate, inconsistent, or non-existent process documentation would significantly increase the process-analysis timeline due to the need for end-to end process analysis.

6 Conclusion

More work is needed to further validate our modeling approach. We have applied our approach to two processes in the security analysis and decision-making domain and submit that it would improve process-modeling efforts in other domains. In addition, there needs to be more research to enable the long-term runtime monitoring of an executable process model in direct support of the real-world application of the process; that is, it is desirable to provide the users of the approach with the capability to execute process models for sufficiently long periods of time such that the modeled processes terminate naturally (i.e., processes that have a definitive end-point or product which causes the process to end) or run indefinitely (e.g., safety processes that involve continuous checking of health and status of the manual and automated functions of a process-control application). Another area of investigation is to extend the tool capabilities supporting our approach, such as to assist process engineers in determining the optimal placement and use of embedded assertions within a statechart-based formal process model.

References

[1] N. Webster and J.L. Mckechnie, *Webster's New Universal Unabridged Dictionary*. second ed., Dorset & Baber, 1983.

[2] D. Drusinsky, *Modeling and Verification Using UML Statecharts: A Working Guide to Reactive System Design, Runtime Monitoring and Execution-Based Model Checking*. Newnes, 2006.

[3] H.A. Gabbar, *Modern Formal Methods and Applications*. Springer, 2006.

[4] J.B. Michael, D. Drusinsky, T.W. Otani, and M.-T. Shing, "Verification and Validation for Trustworthy Software Systems," *IEEE Software*, vol. 28, no. 6, pp. 86-92, Nov./Dec. 2011.

[5] J.F. Monin and M.G. Hinchey, *Understanding Formal Methods*. Springer, 2003.

[6] W. Emmerich and V. Gruhn, "FUNSOFT Nets: A Petri-Net Based Software Process Modeling Language," *Proc. Sixth Int'l Workshop on Software Specification and Design*, pp. 175-184, 1991.

[7] V. Gruhn, "Software Processes Are Social Processes," *Proc. Fifth Int'l Workshop on Computer-Aided Software Eng.*, pp. 196-201, 1992.

[8] V.S. Hibdon and T.C. Hartrum, "An Air Force Organization Process Model Using Formal Software Engineering Techniques," *Proc. Nat'l Aerospace and Electronics Conf.*, vol. 2, pp. 482-489, 1996.

[9] J. Arlow, S. Bandinelli, W. Emmerich, and L. Lavazza, "Fine Grained Process Modelling: An Experiment at British Airways," *Software Process: Improvement Practice*, vol. 3, no. 2, pp. 105-131, 1997.

[10] J. Koehler, G. Tirenni, and S. Kumaran, "From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods," *Proc. Sixth Int'l Enterprise Distributed Object Computing Conf.*, pp. 96-106, 2002.

[11] V. Gruhn and R. Laue, "What Business Process Modelers Can Learn from Programmers," *Science of Computer Programming*, vol. 65, no. 1, pp. 4-13, Mar. 2007.

[12] J. Ye, S. Sun, W. Song, and L. Wen, "Formal Semantics of BPMN Process Models Using YAWL," *Proc. Second Int'l Symp. Intelligent Information Technology Application*, vol. 2, pp. 70-74, 2008.

[13] P.Y.H. Wong and J. Gibbons, "A Process Semantics for BPMN," *Proc. 10th Int'l Conf. Formal Methods Software Eng.*, pp. 355-374, 2008.

[14] J.O. Grady, "Universal Architecture Description Framework," *Systems Eng.*, vol. 12, no. 2, pp. 91-116, May 2009.

[15] Q. Zongyan, P. Liyang, and Y. Hongli, "A Framework for Integrating Human Processes with Business Artifacts," *Proc. IEEE Fifth Int'l Symp. Service Oriented Systems Eng.*, pp. 252-259, 2010.

[16] J.A.H. Alegría, M.C. Bastarrica, and A. Bergel, "AVISPA: Localizing Improvement Opportunities in Software Process Models," Technical Report DCC-2010-6, Univ. of Chile, July 2010.

[17] J.A.H. Alegría, M.C. Bastarrica, and A. Bergel, "Analyzing Software Process Models with AVISPA," *Proc. Int'l Conf. Software and Systems Processing*, pp. 23-32, 2011.

[18] M. Bishop, *Computer Security: Art and Science*. first ed., Addison-Wesley, 2002.

[19] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science Computing Program.*, vol. 8, no. 3, pp. 231-274, June 1987.

[20] Y. Dong and Z. Shensheng, "Modeling Workflow Process Models with Statechart," *Proc. IEEE 10th Int'l Conf. Workshop Eng. Computer-Based Systems*, pp. 55-61, 2003.

[21] D. Drusinsky, M.-T. Shing, and K.A. Demir, "Creation and Validation of Embedded Assertion Statecharts," *Proc. 17th IEEE Int'l Workshop Rapid System Prototyping*, pp. 17-23, 2006.

[22] D. Drusinsky, "From UML Activity Diagrams to Specification Requirements," *Proc. IEEE Int'l Conf. System of Systems Eng.*, 2008.

[23] D. Drusinsky, *Practical UML Based Specification, Validation, and Verification of Mission Critical Software*. first ed., Dog Ear Publishing, LLC, 2011.
[24] M. Sipser, *Introduction to the Theory of Computation*. PWS Publishing, 1997.
[25] J.E. Hopcroft, R. Motwani, and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2007.
[26] D. Kelley, *Automata and Formal Languages: An Introduction*. Prentice Hall, 1995.
[27] D. Drusinsky and M.-T. Shing, "Using UML Statecharts with Knowledge Logic Guards," *Proc. 12th Int'l Conf. Model Driven Eng. Languages and Systems*, pp. 586-590, 2009.
[28] M. Crane and J. Dingel, "UML versus Classical versus Rhapsody Statecharts: Not All Models Are Created Equal," *Software and Systems Modeling*, vol. 6, no. 4, pp. 415-435, 2007.
[29] B. Bruegge and A.H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns and Java*. second ed., Prentice Hall, 2004.
[30] D. Drusinsky, J.B. Michael, T.W. Otani, and M.-T. Shing, "Validating UML Statechart-Based Assertions Libraries for Improved Reliability and Assurance," *Proc. Second Int'l Conf. Secure System Integration Reliability Improvement*, pp. 47-51, 2008.
[31] M.A. Schumann, "Use of Statechart Assertions for Modeling Human-in-the-Loop Security Analysis and Decision-Making Processes," doctoral dissertation, Naval Postgraduate School, June 2012.
[32] C.A.R. Hoare, *Communicating Sequential Processes*. Prentice Hall, 1985.
[33] P.Y.A. Ryan and S.A. Schneider, *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2000.
[34] A. Karniel and Y. Reich, "Formalizing a Workflow-Net Implementation of Design-Structure-Matrix-Based Process Planning for New Product Development," *IEEE Trans. Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol. 41, no. 3, pp. 476-491, May 2011.
[35] W.M. Van Der Aalst and K.M. Van Hee, *Workflow Management: Models, Methods, and Systems*. MIT Press, 2004.
[36] B.F. Van Dongen, W.M.P. Van Der Aalst, and H.M.W. Verbeek, "Verification of EPCs: Using Reduction Rules and Petri Nets," *Proc. 17th Int'l Conf. Advanced Information Systems Eng.*, pp. 372-386, 2005.
[37] J. Clempner, "A Hierarchical Decomposition of Decision Process Petri-Nets for Modeling Complex Systems," *Int'l J. Applied Math. Computer Science*, vol. 20, no. 2, pp. 349-366, June 2010.
[38] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.
[39] IAR Systems, "IAR VisualSTATE," http://www.iar.com/en/Products/IAR-visualSTATE, Jan. 2012.
[40] A. Muelder, "Yakindu," http://www.yakindu.org/yakindu/, June 2011.
[41] M.A. Schumann, "A Statechart Model of the Cross Domain Implementation Process," *Information Assurance Technology Analysis Center Newsletter*, vol. 12, no. 1, pp. 26-30, Spring 2009.
[42] G. Sindre and A.L. Opdahl, "Eliciting Security Requirements by Misuse Cases," *Proc. 37th Int'l Conf. Tech. Object-Oriented Languages and Systems*, pp. 120-131, 2000.
[43] Oxford English Dictionary, "'oracle, n.,'" http://www.oed.com/, Mar. 2012.
[44] M. Schumann and J.B. Michael, "Statechart Based Formal Modeling of Workflow Processes," *Proc. IEEE Int'l Conf. System of Systems Eng.*, 2009.

Michael A. Schumann received the MS degree in systems engineering and the PhD degree in software engineering from the Naval Postgraduate School, Monterey, California, in 2002 and 2012, respectively. Formerly, as a commander in the United States Navy, he served at the Navy Cyber Warfare Development Group in Washington, and as the information warfare commander for the USS Enterprise Carrier Strike Group. Currently, he is the director of the Cyber Research and Solutions Center at KEYW Corporation. His research interests include computer security and the application of formal methods to model human-based processes. He is a member of the IEEE.

Doron Drusinsky received the PhD degree in computer science from the Weizmann Institute of Sciences, Rehovot, Israel, in 1988. He worked for Sony from 1988 until 1993 when he authored BetterState, a UML statecharts design tool that was later acquired by ISI/WindRiver Systems. He authored the Temporal Rover and StateRover V&V tools currently in active use by the NASA IV&V Facility. Currently, he is an associate professor with the Department of Computer Science, Naval Postgraduate School, Monterey, California, and the president of Time-Rover. He has written many technical papers on the subject and published in 2011 the book titled *Practical UML-Based Specification, Validation, and Verification of Mission-Critical Software*. His research interests include computer-aided specification and verification and validation of safety-critical systems.

James B. Michael (S'87, M'92, SM'97) received the PhD degree in information technology from George Mason University, Fairfax, Virginia, in 1993. He was formerly an assistant researcher engineer with the University of California at Berkeley. Currently, he is a professor with the Departments of Computer Science and Electrical & Computer Engineering, Naval Postgraduate School, Arlington, Virginia. His research interests include application of formal methods for verification and validation, reliability and safety engineering, and computer security and digital forensics. He received the 2010 Engineer of the Year Award from the IEEE Reliability Society for his contributions to the field of trustworthy distributed systems. In 2013, he received the Department of the Navy Civilian Meritorious Service Award. He is a senior member of the IEEE.

Duminda Wijesekera received the PhD degree in mathematics from Cornell University in 1990 and the PhD degree in computer science from the University of Minnesota in 1997. He has worked as a senior systems engineer with Honeywell Space Systems, Clearwater, Florida, as a visiting postdoctoral fellow at the University of Minnesota's Army High Performance Computing Research Center, Minneapolis, Minnesota, and as an assistant professor of mathematics and computer science, University of Wisconsin, Superior, Wisconsin. He is currently a professor with the Department of Computer Science at George Mason University, Fairfax, Virginia. His research interests include the application of formal methods to the analysis of computer security and other types of protocols and various topics in theoretical computer science on logic. He is a senior member of the IEEE.