Using the Discrete Event System Specification to model Quantum Key Distribution system components

Jeffrey D Morris, Michael R Grimaila, Douglas D Hodson, Colin V McLaughlin and David R Jacques

The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology published online 17 October 2014
DOI: 10.1177/1548512914554404

Applications

JDMS

Journal of Defense Modeling and Simulation: Applications, Methodology, Technology
1–24

DOI: 10.1177/1548512914554404
dms.sagepub.com
SAGE

Using the Discrete Event System Specification to model Quantum Key Distribution system components

Jeffrey D Morris[1], Michael R Grimaila[1], Douglas D Hodson[1], Colin V McLaughlin[2] and David R Jacques[1]

Abstract

In this paper, we present modeling a Quantum Key Distribution (QKD) system with its components using the Discrete Event System Specification (DEVS) formalism. The DEVS formalism assures the developed component models are composable and exhibit well-defined temporal behavior independent of the simulation environment. These attributes enable users to assemble a valid simulation using any collection of compatible components to represent complete QKD system architectures. To illustrate the approach, we introduce a prototypical "prepare and measure" QKD system, decompose one of its subsystems, and present the detailed modeling of the subsystem using the DEVS formalism. The developed models are provably composable and exhibit behavior suitable for the intended analytic purpose, thus improving the validity of the simulation. Finally, we examine issues identified during the verification of the conceptual DEVS model and discuss the impact of these findings on implementing a hybrid QKD simulation framework.

Keywords

Conceptual modeling, Discrete Event Simulation, Discrete Event System Specification, modeling and simulation, Quantum Key Distribution

1. Introduction

Cryptography, the practice and study of techniques for securing communications between two authorized parties in the presence of one or more unauthorized parties, is the centerpiece of a centuries old battle between code maker and code breaker.[1] Historically, only financial, government, and military applications used cryptography, but today much of modern society depends on cryptography to provide security services including confidentiality, integrity, authentication, and non-repudiation.[2] While there are many types of cryptography, only the One-Time-Pad (OTP) symmetric key algorithm is ''information-theoretically secure''.[3,4] All other forms of cryptography are breakable if the adversary has enough cipher text, computational resources, and time.[5] Despite its strength, the OTP is not in common use because of its requirement that keys are random, equal in length to the message, and are never reused. These requirements impose significant limitations on use of the OTP in most applications due to the costs involved with secure key generation and distribution.

Quantum Key Distribution (QKD) is a technology that offers the means for two geographically separated parties to create a shared secret key.[6] QKD is unique in its ability to detect any third-party eavesdropping on the key exchange, assuring the secrecy of the key. This is possible due to the fundamental laws of quantum mechanics, which ensures any third-party eavesdropping on the quantum channel introduces detectable errors. Combining a QKD-

[1]United States Air Force Institute of Technology, Wright-Patterson AFB, USA
[2]Naval Research Laboratory, USA

Corresponding author:
Michael R Grimaila, United States Air Force Institute of Technology, AFIT/ENV; 2950 Hobson Way, Wright-Patterson AFB, OH 45433-7765, USA.
Email: Michael.Grimaila@us.af.mil

generated key with the classical OTP realizes an ''unconditionally secure'' cryptosystem that provides significant benefits in military applications. QKD-secured systems could connect command and control nodes throughout the communication channels, connecting commanders with their leadership through terrestrial and ground to space circuits. Satellites with onboard QKD devices could communicate globally, and distribute new keys to friendly systems located anywhere there is line of sight to the platforms. Changing the keys several times a second could make it virtually impossible for cyber adversaries to decrypt the communications traffic. Commercial QKD systems are available today and several governments have already instituted QKD to secure communication circuits.[7,8]

1.1 The need for QKD simulation

QKD technologies have not been thoroughly studied from a systems-level perspective. Real-world QKD systems are constructed from non-ideal components that differ, sometimes significantly, from the ideal components specified during the original conceptual system design. Therefore, there is a need to develop an efficient integrated modeling and simulation (M&S) capability to understand the impact non-ideal components have on the performance and security of different QKD system architectures. The capability to thoroughly study QKD systems is a prerequisite before these systems can be evaluated and be accepted in military applications. There exist few QKD simulations beyond those that model specific hardware or situations. An example is the Austrian Institute of Technology's AIT QKD Software project[9] that attempts to model an entire QKD network but is based mainly on their entanglement QKD hardware. An extensive literature search over several years revealed no other system-level QKD M&S efforts.

To address this shortcoming, we have developed a modular simulation framework, named *qkdX*, which provides users with the capability to model rapidly, simulate, and study QKD system architectures. This simulation capability provides hybrid functionality as it abstracts continuous time QKD system signals (e.g., electrical signals and optical pulses) into a representation suitable to be transported as events in a Discrete Event Simulation (DES) environment.[10] These abstract models of continuous time signals are reconstructed as necessary to facilitate continuous time operations including integration and interference calculations. A continuous time simulation of a complete QKD system is infeasible, due to the enormous number of optical pulses generated during system operation, and necessitates the use of DES. The design features of the *qkdX* framework include a hybrid discrete-continuous modeling approach to more accurately capture quantum effects; a modular design to allow quick and efficient changes to the system under study; parameterized components allowing for multiple varying instances; a provably composable system allowing for hierarchal construction of complex systems from simple components. The framework supports multiple qubit encoding schemes (i.e., polarization-based, phase-based, and entanglement), multiple protocols (e.g., BB84, SARG04, E92), and various QKD applications (e.g., buried optical fiber, terrestrial directional free-space optical link, and multiplexed transmissions).

The researchers, in consultation with Subject Matter Experts (SMEs) in the optical physics and electrical engineering domains, determined the abstraction necessary for each signal model. The abstraction enables a system-level simulation where signals propagate through the system as discrete events, but can be reconstructed into a continuous time representation when mathematical operations or transformations of the signals are required. The details of the optical pulse model and related mathematical transforms are outside the scope of this paper. Instead, we focus on the proper modeling of the temporal behavior and internal ''state'' of QKD system components. To capture this temporal behavior and the state of components, we use the Discrete Event System Specification (DEVS).[11] In the past, DEVS has been used to model high-level architectures, hybrid systems, cell spaces, distributed supply chains, test and evaluation, forest fires, environmental systems, building performance models, and other problem spaces.[11–18] This paper presents, to our knowledge, the first use of DEVS to model QKD system components.

1.2 The need for validity in simulation

Model validity is a necessary condition for the credibility of simulation results.[19] Model validation, according to Balci,[20] is ''substantiating that the simulation model, within its domain of applicability, behaves with satisfactory accuracy consistent with the study objectives''. Model validation is the comparison of model behavior to the behavior of the system under study when both are responding to identical input conditions.[21] Model testing identifies failures, corrects them, and then retests to the required accuracy and behavior.[19,20]

Complete testing of a model throughout its solution space is not possible. Such testing is cost and time prohibitive; instead, testing continues until attaining sufficient confidence in the model for its intended purpose.[21]Figure 1 shows the relationship between value, cost, and model confidence. As user confidence in the model increases, there is a corresponding logarithmic increase in the value of the model, but the cost increases exponentially. Eventually the gain in value is negligible but costs continue to increase steeply.

Validity and model confidence relate closely. The better the belief the model accurately represents the system

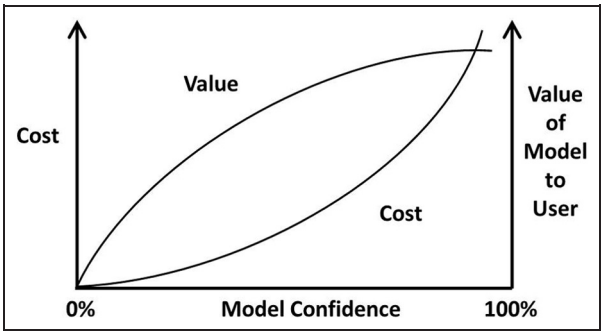

Figure 1. Model confidence.[21]

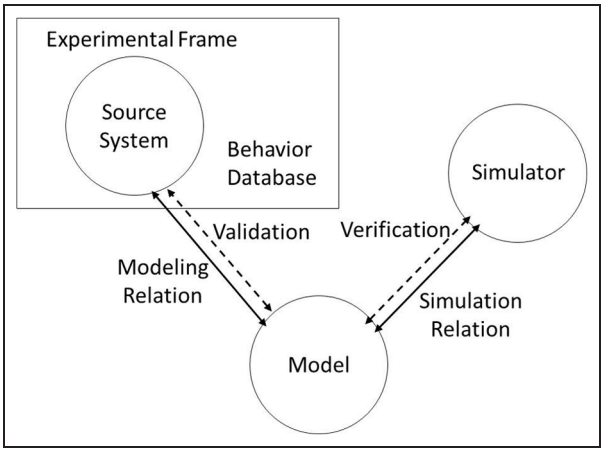

Figure 2. Basic entities in modeling and simulation, adapted from Zeigler et al.[28]

under study, the higher the validity of the model. Higher validity suggests a higher confidence in the model being useful for its purpose. Exhaustive testing brings higher confidence, but at much greater cost. Our research is focused upon exploring a means to increase model validity without the need to exhaustively test the entire solution space. Specifically, this research focuses on how using the DEVS formalism and conceptual model validity theory increases the validity of a QKD system-level simulation. To achieve this goal, we present modeling QKD system components using the DEVS formalism.[22] Representing component behavior using the DEVS formalism ensures the developed conceptual models exhibit composability and deterministic temporal behavior independent of the simulation environment. In addition, we identified unforeseen benefits that arise when using a strict modeling formalism.

The remainder of this paper is organized as follows. Section 2 reviews the DEVS formalism and conceptual modeling. Section 3 explains the use of DEVS in our modeling effort. We introduce a prototypical QKD system architecture and decompose its ''classical pulse generator'' (CPG) subsystem in Section 4. Section 5 presents modeling the CPG subsystem using DEVS atomic and coupled models. Section 6 presents our findings, reviews issues identified during simulation and verification of the conceptual model, and discusses the impact of these findings on implementing the hybrid simulation framework. Finally, we provide concluding remarks in Section 7.

2. Discrete Event System Specification and conceptual model validity

2.1 What is DEVS?

Zeigler first proposed the DEVS in 1976 as a hierarchal formalism for decomposing complex discrete event systems into simple components, leading to *well-defined* behavior of the overall model.[23] Zeigler states ''…the set of all dynamic systems is taken as a well-defined class in which each system has a set of input time segments, states, state transitions and output time segments.'' DEVS interprets the dynamic systems as sets and functions and sets conditions needed for a well-defined specification.[24] DEVS defines system behavior, syntax, and structure, enabling modularity within a DES by building complex systems from simple (*atomic*) components. It uses dynamical systems theory as a means to canonically represent system behavior and provide provable *closure under coupling* (also known as *composability*).[25,26] DEVS provides a structured way to represent complex systems in a hierarchal manner to create *coupled* (compound) modules, or subsystems, to create complete system models. The theory states ''…the dynamic system specified by a coupled model can be represented as (more technically, is behaviorally equivalent to) an atomic DEVS system''.[24]

DEVS separates the model from the source system and the simulator. This allows for a conceptual model not tied to any particular simulator and creates a bounded source with finite inputs and outputs. Hoffman describes DEVS as a ''theoretical confirmation'' of transformations between different techniques and tools for modeling systems.[27]Figure 2 shows a representation of basic entities in M&S. The source system (i.e., the system under study) couples with a database of behaviors derived from a set of inputs. This *experimental frame* defines the system of interest the modeler is trying to capture[28] and allows the modeler to create a conceptual model for the system under study. Zeigler links the experimental frame and the model with a *modeling relation* and the model and the simulation with a corresponding *simulation relation*. The first relation describes how well-observed system behavior matches model-generated behavior (validity) and the second with how well the simulation executes model instructions

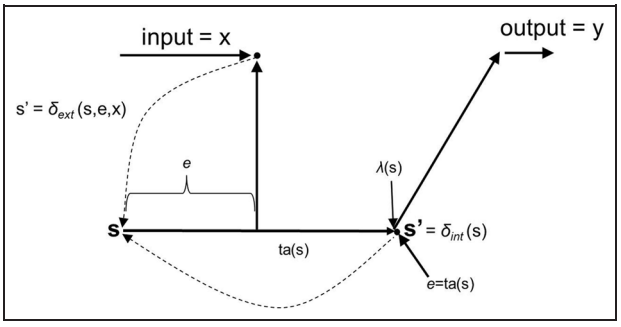

Figure 3. Discrete Event System Specification (DEVS) sequence diagram, derived from Zeigler et al.[30]

(verification). Note the model is the bridge between the system and the simulator.

DEVS provides a concise way of describing the inputs, states, outputs, and timing of a system under study. It is a formal language used to define the conceptual model of the system.[29]

A DEVS atomic model has sets of inputs, states, and outputs along with transition and output functions to construct a representation of any dynamic system.[24]Figure 3 provides a graphic representation of DEVS modeling state transitions in response to incoming events. We start in an initial state s and remain in that state for some time advance period ta. Once ta is reached, the system may output some value y per the output function $\lambda(s)$. Immediately after the output, the system goes through the internal transition, δ_{int} (s), based on the current state. The system changes to state s', which becomes the new state s and the cycle starts over. If the state receives an external disturbance during the time ta, at some elapsed time since the last transition, e, the system undergoes an external transition δ_{ext} (s,e,x). The external transition uses the existing state, s, the time elapsed, e, and the input values, x, to determine the new state, s', and the cycle starts anew.

While there are different types of DEVS, Parallel-DEVS has several characteristics necessary to model QKD components. The Parallel-DEVS formalism specifies the following about each atomic model:[31]

- ports between models are represented explicitly – there can be any number of input and output ports;
- atomic DEVS models can handle *bags* of inputs and outputs;
- a bag can contain many elements with possibly multiple occurrences of its elements;
- the external transition function handles inputs of bags;
- the output function can generate a bag of outputs;
- the confluent transition function, $\delta_{con}(s,\ ta(s),\ x)$ decides the processing order of simultaneous external and internal events.

In this paper, we make use of Parallel-DEVS as it provides the unique abilities of queues, necessary to handle multiple arriving optical packets, and the confluence transition function, necessary for handling simultaneous events.[32]

2.2 Why use DEVS?

The DEVS formalism ensures well-defined component temporal behavior and provable closure under coupling (i.e., composability), allowing for easier verification of correctness of component compositions, and improving the validity of system representations. Thus, for our purpose, we are ensured the temporal behavior of high-level QKD system representations reflects the dynamics of its constituent parts. In other words, the burden of verifying high-level system dynamics focuses on correct modeling of constituent parts (components); once accomplished, we can assemble high-level representations for study with confidence the assembly process itself has not introduced unforeseen effects or anomalies.

2.2.1 Conceptual modeling and validity. This research focuses on the use of DEVS conceptual modeling to capture the behavior of components found in the optical path of a QKD system. The question remains: Why do this? Conceptual modeling is the process of determining what to model to be useful.[33] This process has been described as the conceptualization of real-world referents that varies from modeler to modeler,[27] whereas Robinson[34] describes the conceptual model as ''non-software specific description of the simulation model that is to be developed''. Deciding the appropriate ''wrongness'' (abstraction), agreement on the model, and model validation are some of the objectives of conceptual modeling.[32] Although conceptual modeling has been described by some as more art than science,[35] Robinson[34] provides a framework for conceptual modeling and lists five activities in a process for conceptual modeling:

- understanding the problem situation;
- determining the modeling and general project objectives;
- identifying the model outputs (responses);
- identifying the model inputs (experimental factors);
- determining the model content (scope and level of detail), identifying any assumptions and simplifications.

This is where the usefulness of the DEVS formalism becomes apparent. It provides a mathematically proven process to work through the objectives. Using DEVS to model the components for the QKD demonstration architecture gives a way to meet the objectives of conceptual modeling while accomplishing the steps in the modeling process. DEVS forces the modeler to have a deep understanding of behavior and timing, which in turn requires understanding the problem, the modeling objectives, and capturing inputs and outputs.

How hard it is to distinguish between the model and the source system is the question of *validity*.[36] The harder it is to distinguish between the model and the source system in the experimental frame, the greater the validity. Note that a model's validity only applies to the experimental frame of interest. Change the frame and you change the validity of the corresponding model.

As mentioned, it is cost-prohibitive to check every possible model combination and trying to validate the entire model space by model-checking or theorem-proving approaches is nearly impossible once a model has many connections or interactions.[37] DEVS allows for increased model validity without having to check the entire model space.

2.2.2 How does DEVS increase validity?. Since validity is a measure of ''closeness,'' how does DEVS increase validity? Using DEVS forces allows the modeler to have a deep understanding of the modeled behavior because the formalism requires it. This lessens or eliminates undesired, unexpected, or emergent behavior. By knowing exactly how the model behaves, it can be matched and changed to the observed behavior of the source system.

DEVS provides three levels of validity for conceptual models: replicative, predictive, and structural.[36] Each level of validity meets the requirements of the previous level(s). The model and system achieve replicative validity if their behaviors agree to acceptable levels for all experiments captured in the behavioral database for the experimental frame. The second, predictive, requires the model to generate the same output as the system for any experiment not captured in the experimental frame database. This requires the model to be able to be set into the same state as the system for the experiment, for any acceptable starting state. Finally, structural validity requires the model and system to have a corresponding step-by-step, component-by-component transition through all possible states. Any model properly using DEVS achieves all three of these states, making it harder to distinguish between the model and system under study, and increasing its validity, per the earlier definition.

Another consideration is the temporal behavior of the source system. In many discrete event simulators, the underlying implementation of the simulator influences the behavior of the model when multiple simultaneous events occur. DEVS addresses this problem by providing a confluence function to express the behavior of the model for these situations. This means a DEVS model exhibits the same behavior on any DEVS-compliant simulator and if the simulation relation is sound, the behavior can be replicated on any DES.

Lastly, DEVS promotes sound model development when used as an intermediate step towards developing a large DES model using a non-DEVS-compliant simulator. For example, some discrete event simulators schedule events in the future for convenience.[38] This situation can cause problems; DEVS avoids this as there are no future events in the formalism. There are only two types of time in DEVS: the time advance (*ta*) and the elapsed time (*e*). This forces the modeler to carefully consider time and input interaction on all states.

3. Using Discrete Event System Specification to model Quantum Key Distribution system components

3.1 Methodology

3.1.1 The modeling process. The QKD modeling process began with discussions between the research team and the SMEs in the areas of optical physics, quantum physics, electrical and software engineering. These discussions led to an agreement on simulation objectives, what needs modeling, the fidelity necessary, assumptions, and simplifications needed or wanted within the model. The results were a small-scale proof-of-concept simulation using the OMNeT++ DES[38] to show the basic premises were sound, selecting DEVS to build the conceptual model, and continue using OMNeT++ as the platform for the full QKD simulation.[10]

The research effort was divided between three sections of the team: domain engineers, M&S engineers, and platform engineers.[39] Domain engineers were researchers from the Laboratory of Telecommunication Sciences and the Naval Research Laboratory, providing expertise in the fields of optical and quantum physics. The primary author of this paper filled the role of the M&S engineer, while the faculty of the Air Force Institute of Technology provided expertise in electrical and software engineering. The three groups came together to form the collaborative research team for the *qkdX* simulation framework.

The optical physics SME created a mathematical model for each of the optical components that captured parameters and behavior believed necessary for the model. Model creation used a combination of data measured during laboratory experiments in conjunction with component data sheets and existing reference literature. Creating and

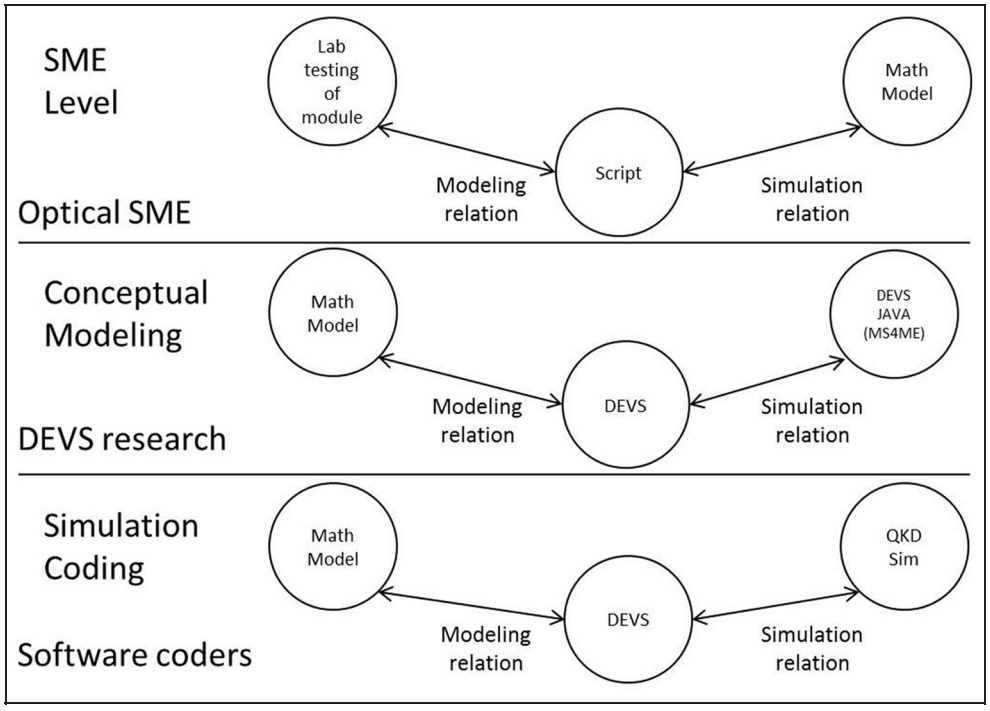

Figure 4. Levels of modeling and simulation.

verifying the correctness of the developed math models used Mathematica, a well-known math computation software package.[40]

The next step was to transform the mathematical model into a DEVS pseudocode model. These mathematical models become the ''source system'' shown in Figure 2 and, consequently, the basis for the QKD simulator. The modeler reviewed the math models to understand the necessary transformation functions, reviewed quantum and optical physics literature and consulted with the SMEs to understand the required component behavior. Product literature for existing physical components provided additional information for acceptable component input and output ranges. The DEVS models captured this information using phases, states, and transitions and submitted the component models back to the optical SME for review.

Once complete, the DEVS pseudocode became the basis for creating the model in a DEVS-compliant simulator, MS4ME.[41] MS4ME is a product of RTSync (www.rtsync.com), a spin-off from the Arizona Center of Integrative Modeling and Simulation (ACIMS).[42] MS4ME provides a structured user interface for modeling built on top of the DEVSJAVA simulator.[43] For each component, the output from the MS4ME simulator was compared against the expected behavior of the DEVS model. This modeling was a check on the DEVS pseudocode and ensured the models met the requirements of the formalism and captured the appropriate behavior. Once checked, the DEVS pseudocode became the basis for the simulation modelers to create the *qkdX* framework. As shown in Figure 4, DEVS is the intermediate step between the SME mathematical model and the QKD simulation.

3.1.2 SME to conceptual model to simulation cycle. During the conceptual modeling process, the modeler worked with the software and electrical engineers to capture the hardware and software behavior of QKD devices. A constant review process looked for differences between the proof-of-concept demonstrator and the detailed DEVS models. Once complete, the DEVS models went to the simulation modelers (the platform engineers) for use in adapting the existing proof-of-concept simulation code to agree with the conceptual model. This process is an example of the simulation relation.

The research team held weekly teleconferences and had several site visits in a continuing effort to better understand and model the QKD system. Development of the DEVS modules and translation into both the MS4ME DEVS simulator and the *qkdX* simulation framework resulted in the identification of multiple inconsistencies between the representations. These inconsistencies were reconciled to yield canonical behavior between all simulation models. Throughout this process, the modeler continued to consult with the optics SMEs. Each completed model underwent review by the SMEs to ensure the DEVS model captured

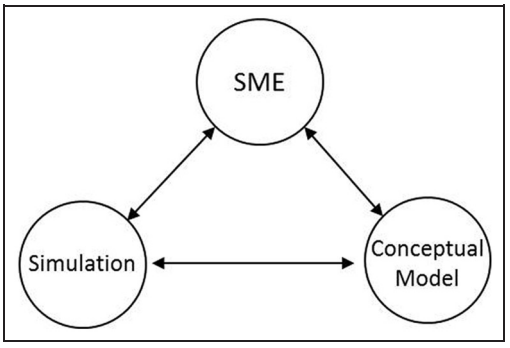

Figure 5. Research modeling process.

the proper behavior and essential parameters. This is an example of the idea of the model relation.

Figure 5 shows an overview of our research modeling process. The SME generated the mathematical models used to create the conceptual models. Constant two-way communication involved the SME and the simulation modelers in the conceptual modeling process. Once acceptable to the SME, the conceptual models were given to the simulation modelers for translation into the simulation framework. Once again, there was constant communication between the conceptual modeler, the SME, and the framework modelers. This circular modeling process ensured acceptance between all parties.

3.1.3 Experimental frame. The experimental frame built for testing the DEVS models closely matches the concepts put forth by Traore and Muzy,[44] where a frame was built to ''surround'' each model and provided inputs and accepted the model outputs, as shown in Figure 6. The frame has an ''Upstream'' piece as a Generator, accepting the start/stop commands and feeding optical, environmental, and control data to the component under test. The component output its values to the ''Downstream'', which worked as the Transducer, and the frame had internal logic that checked the outputs (acting as the Acceptor).

Once the simulation is started, the Upstream injects a combination of optical, environmental, and control messages to the model, which processes these and outputs a combination of optical and control messages. These optical models never output environmental messages, as these messages are receive-only messages. The Downstream receives the model outputs and provides the execution times of status changes within the model. Finally, the experimental frame has internal logic to check the model for consistency. This experimental frame construct allowed for easy swapping of the models under test and the same system was used to test the coupled models discussed later in this paper.

4. A prototypical polarization-based BB84 prepare and measure Quantum Key Distribution system

Consider the model of a prototypical QKD system that uses a polarization-based, Bennett and Brassard[45]''BB84'' prepare and measure protocol, shown in Figure 7. The QKD system comprises an ''Alice'' subsystem, a ''Bob'' subsystem, an authenticated public communications channel, and a quantum communication channel. Due to the complexity of a QKD system, we focus our discussion on decomposition within the Alice subsystem, the Alice quantum module CPG subsystem, to illustrate modeling QKD system components using DEVS.

The Alice subsystem responsibilities include producing and encoding photons with candidate secret key bits and sending the photons to the Bob subsystem via the quantum channel. The Bob subsystem receives the encoded photons and decodes them to recover the candidate key bits. Alice and Bob coordinate their system operations by communicating over the authenticated public channel.

4.1 Alice subsystem decomposition

The Alice subsystem contains several subsystems, including a system controller module, a public channel module, a dedicated QKD module, a quantum module, a clock, and a True Random Number Generator (TRNG), as shown in Figure 8.

The Alice system controller module is responsible for controlling the Alice subsystem and serves as the master controller to coordinate operations between Alice and Bob. The public channel module interfaces with the system controller module and provides connectivity to the remote system via the public channel. The dedicated QKD module controls QKD-specific processing such as error detection and correction, sifting, and privacy amplification. The quantum module is responsible for generating the quantum state in optical pulses before sending them to Bob via the quantum channel. The clock source provides reference timing for all synchronous devices. Since the security of a QKD system is a strong function of the randomness of the numbers it generates, a TRNG such as the idQuantique Quantis optical random number generator is typically used to provide the required source of entropy.[46]

4.2 Alice quantum module subsystem decomposition

The quantum module decomposes into nine different subsystems, Table 1 shows a brief description of each subsystem function and Figure 9 illustrates the decomposition.

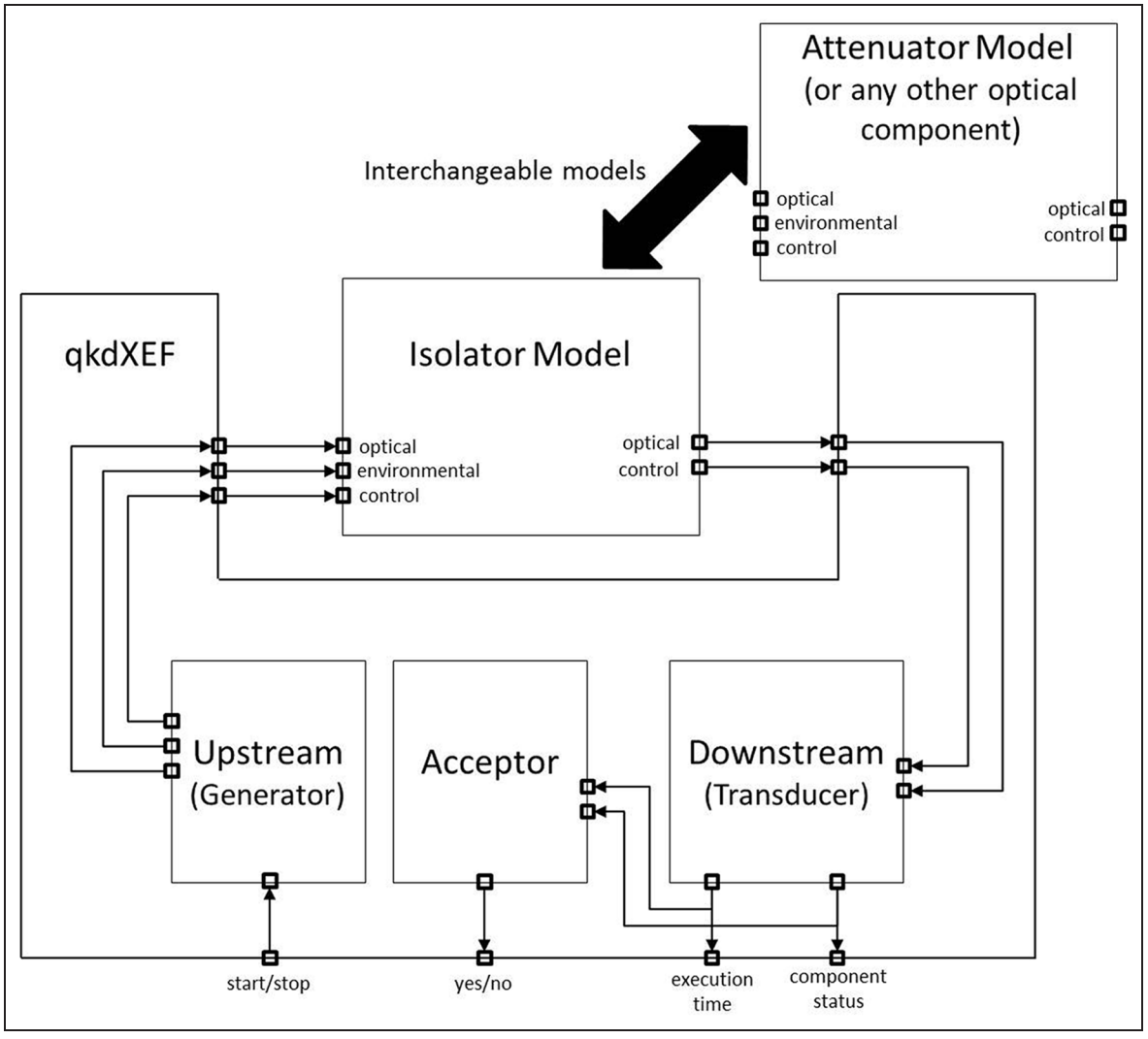

Figure 6. *qkdX* experimental frame model.

4.3 Alice classical pulse generator subsystem decomposition

The ideal conceptual model of a QKD system specifies polarization-encoded single photons with the desired bit and basis. In reality, reliable on-demand single photon pulse generators are an unrealized technology. Real-world QKD system implementations instead generate a laser pulse containing millions of photons and strongly attenuate the pulse down to statistical sub-photon (quantum) levels. Within the Alice quantum module, the CPG subsystem generates the laser pulses and shifts them into a known polarization. The CPG subsystem contains the components shown in Figure 10.

The CPG subsystem contains a controller, a laser, an isolator, an optical polarizer, an optical bandpass filter, a beamsplitter, a classical detector, electrical interfaces, and interconnecting polarization-maintaining (PM) optical fiber. We first discuss the behavior of the subsystem as a whole and then briefly discuss the behavior of each of the components contained within the CPG.

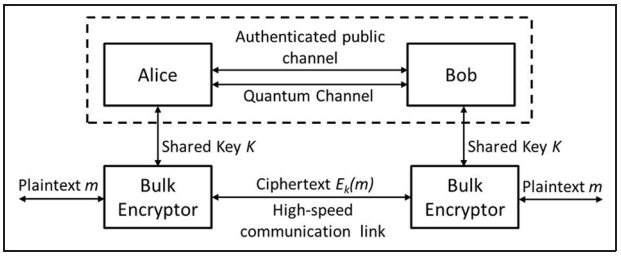

Figure 7. Quantum Key Distribution (QKD) context diagram showing the QKD system and the bulk encryptors using the generated key.

Table 1. Description of Alice quantum module subsystems.

Subsystem	Function
Classical pulse generator (CPG)	Generates a multi-photon pulse
Polarization modulator	Polarizes the photon pulse into the desired polarization
Electronically variable optical attenuator (EVOA)	Creates decoy states to mitigate photon splitting attacks
Fixed Attenuator	Converts classical laser pulses to quantum levels by attenuating to weak-coherent levels
Optical security layer	Detects optical probing attacks
Wave division multiplexer (WDM)	Multiplexes signal and timing pulses
Timing pulse generator	Generates a timing pulse used for synchronization
Switch	Allows generated pulses to be directed to the Output Power Monitor for loop-back testing
Output power monitor	Monitors the output optical power

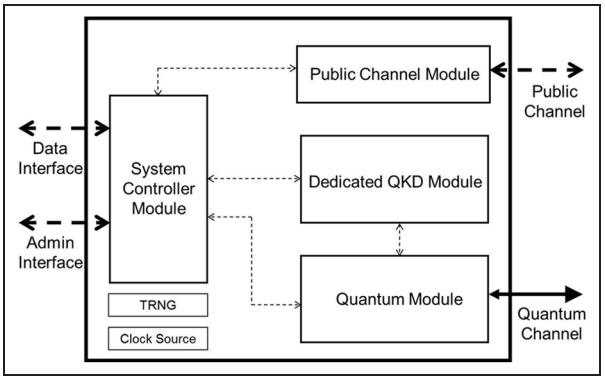

Figure 8. Alice subsystem decomposition.

4.4 Individual CPG component behavior

4.4.1 CPG controller. The controller is an electrical device containing digital and analog circuits responsible for controlling the laser and monitoring the classical detector. It has a bidirectional electrical interface to the quantum module controller, an electrical output to the laser, and an electrical input from the classical detector. It receives commands from the quantum model controller, sends fire commands to the laser, and monitors the health of the laser.

4.4.2 Laser. The laser is an electro-optical device that contains an optical oscillator and emits coherent light. It has an electrical input to receive control messages and an optical output to emit generated pulses. Within the simulation, the laser creates optical pulses when it receives a ''fire'' command from the controller. The laser generates short-duration laser pulses (e.g., 1 mW peak intensity with a 500 ps duration) containing millions of photons. The output of the laser couples to the input of the isolator via PM fiber.

4.4.3 Isolator. The isolator is an optical device with two bidirectional optical ports that passes light in the forward direction while significantly attenuating light moving in the opposite direction. Optical signals arriving at one port propagate to the other port after a defined propagation delay with the attenuation based on the propagation direction. The isolator assures that virtually no light (e.g., reflections or light from external sources) enters the laser. The output of the isolator is coupled to the input of the polarizer via PM fiber

4.4.4 Polarizer. The polarizer is an optical device with two bidirectional optical ports allowing light of one

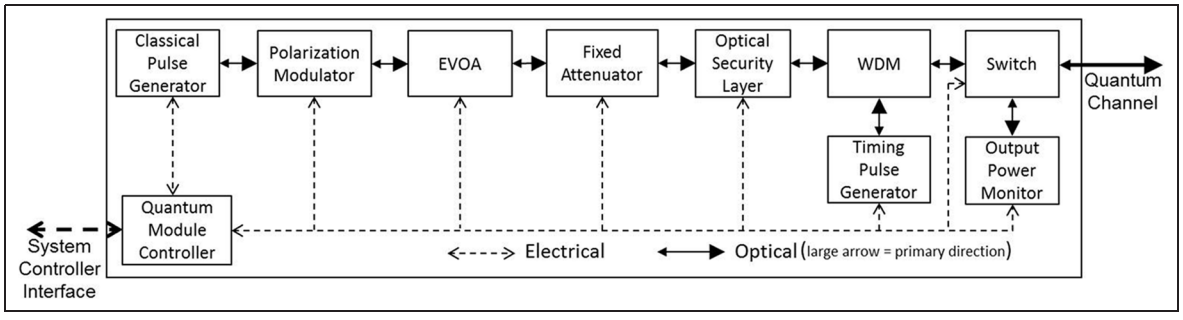

Figure 9. Alice quantum module subsystem decomposition showing internal subsystems and components.

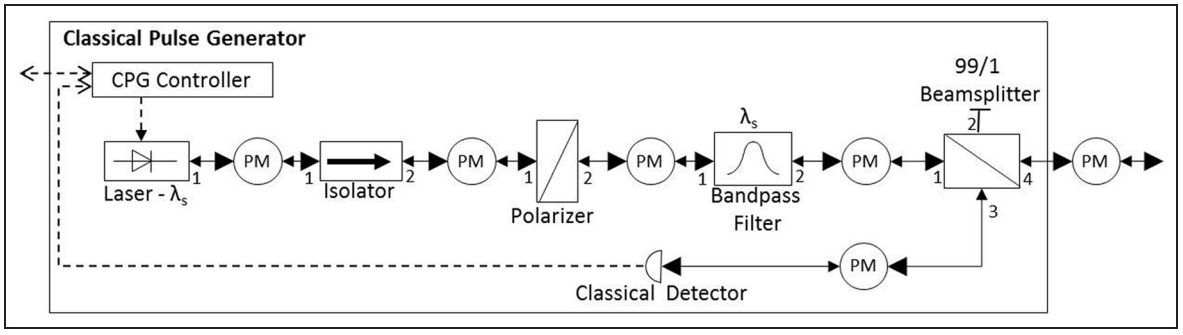

Figure 10. Classical pulse generator subsystem showing internal components.

polarization to pass while highly attenuating light orthogonal to the passed light. Optical signals arriving at one port propagate to the other port after a defined propagation delay and are polarized depending on the polarizer orientation with respect to the connected fiber. The output of the polarizer is coupled to the input of the optical bandpass filter via PM fiber.

4.4.5 Bandpass Filter. The bandpass filter is an optical device with two bidirectional optical ports that passes the optical energy in a narrow band around the signal wavelength, λ_S, but strongly attenuates other wavelengths. This ensures that only the appropriate signal wavelength leaves the subsystem while preventing other sources of light from entering the laser. Optical signals arriving at one port propagate to the other port after a defined propagation delay and are attenuated based on the wavelength of the signal. The bandpass filter output couples to port 1 of the beamsplitter.

4.4.6 Beamsplitter. The beamsplitter is an optical device used to split a single beam of light into two components. It can also be used to combine two beams of light into one stream. Unlike most of the optical devices, it has four bidirectional optical ports. In the splitting configuration, optical signals arriving at one port are split into two beams, propagating to the appropriate output ports after a defined propagation delay. Common splitting ratios are 50:50, 90:10, and 99:1, but devices exist in almost any ratio. Beams can also be split according to optical wavelength or polarization.

The beamsplitter passes 99% of the pulse through to port 4, leaving the CPG and connecting to the next quantum module subsystem, as shown in Figure 10. Meanwhile, port 3 passes 1% of the pulse on to the classical detector via PM fiber.

4.4.7 Classical detector. The classical detector is an opto-electrical device containing an optical photodiode and support electronics to generate an electrical signal proportional to the power contained in the optical pulse. This signal connects to the controller, which stores this information and checks to see if it falls below a predefined threshold. If so, the controller notifies the quantum module controller of an error condition.

4.4.8 Polarization-maintaining optical fiber. PM fiber is an optical component used to interconnect optical devices. It has two bidirectional optical ports. Optical signals arriving at one port propagate to the other port after a defined propagation delay. Attenuation is a function of the type and the length of the fiber. PM fiber maintains the polarization of optical signals injected along its fast and slow axes.

5. Modeling the Alice classical pulse generator subsystem using the Discrete Event System Specification formalism

In this section, we discuss features common to all components in the quantum optical path, DEVS modeling of the isolator, and the CPG. We selected the CPG as it is unique in containing many types of components found in QKD system simulation: electrical, optical, electro-optical, and opto-electrical.

5.1 Common behaviors of optical components

All the components that interact with optical signals share some common traits. These include component state, losses to optical intensity, deleting weak packets, environmental ports, and handling multiple pulses. The following section explains these commonalities.

Each modeled optical component is in one of three states: normal, degraded, or damaged, as shown in Figure 11. In the normal state, the component uses a mathematical transform to generate the resulting normal output pulse(s) of the component under normal conditions. When in the degraded state, the component temporarily uses a

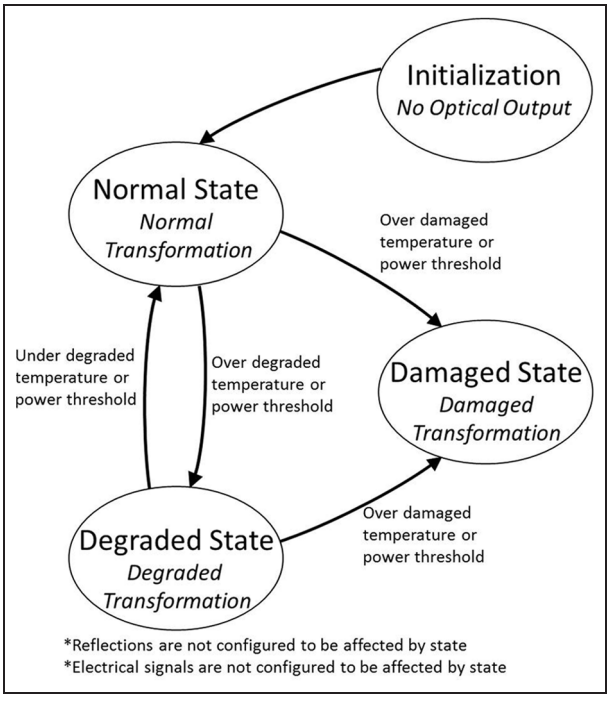

Figure 11. Optical component state transition diagram showing initialization and three states.

different transform to generate the resulting degraded output pulse(s), but returns to normal after a predefined time period or condition (e.g., when temperature returns to a normal level). When in the damaged state, the component permanently uses a transform to generate the resulting damaged output pulse(s), if any.

Components change states based on the entering optical packet power and from the ambient temperature of the component. If the power in an optical pulse exceeds the defined degraded optical power threshold, it will temporarily enter the degraded state. Similarly, if the power in the optical pulse exceeds the defined damage optical power threshold, it will permanently enter the damaged state. The ambient temperature can also result in a component entering a degraded or damaged state.

When an optical signal arrives at an optical component, a small portion of the light reflects opposite to the light propagation direction. The *return loss* parameter of the component determines the reflected amount, depending on different experimental frames. Certain simulation studies may require the capability to accurately represent reflections, while it is not desired in other studies. Therefore, we added the capability to turn reflections on and off for each component, reducing the number of events when not needing this fidelity. The pulse not only loses intensity to reflections, but a small amount is lost when entering the component. This *insertion loss* parameter may also be turned off and on.

As the optical pulses suffer losses propagating through the system, they are deleted when the optical power drops below a defined minimum threshold. This reduces the number of events and prevents an infinite number of reflections bouncing between two reflecting components. Since fiber couples the optical components together, we chose to implement this function in the fiber.

When dealing with a series of single pulses, as is the case in the normal operation of a QKD system, an optical component typically will have a single pulse propagating through it. However, a robust simulation framework must allow components to be able to handle multiple optical pulses simultaneously arriving and/or propagating through it. Therefore, optical components need a queue to store the multiple pulses. The queue contains port-value pairs and any metadata required to process the pulse. At a minimum, the metadata consists of the arrival port and the time remaining before the transformed optical pulse propagates out of the component. As time transpires in the model, a timer ensures the next transformed pulse processes at the appropriate time and, based on the current component state, the appropriate transform generates the output pulse.

Finally, each component has a separate environmental port so the simulation controller can send environmental messages to mimic temperature variations or physical perturbations (e.g., vibration) in the system. The temperature state has no effect on reflections: a component will reflect optical packets regardless of its current state.

5.2 Basic design of a DEVS model for the isolator

5.2.1 Isolator phase transition diagram. In this section, we discuss how to model the isolator component using the DEVS formalism. The isolator is representative of most simple optical components and shares their basic behaviors. The isolator allows light to pass in the forward direction while significantly attenuating light moving in the opposite direction. External, internal, confluence, output, and time advance functions represent the device, as with all DEVS models.[47] These functions contain logic governing how the component responds to inputs and what and when it will output. DEVS uses a phase within a state as a ''marker'' to keep track of internal functions within the state. Figure 12 shows the DEVS phase transition diagram for the isolator and shows the phases (rectangles), the transition arcs (arrows), and any notes for the component.

The isolator phase transition diagram shows the full state description at the top, which includes the current phase (''Passive'', ''Reflect'', ''Respond''), the current time advance (σ), variables with names *store, temperature, overtemp, overpower, interruptRespond* and the port-value pairs in the *queue*. Taken together, these variables allow construction of the full state of the isolator.

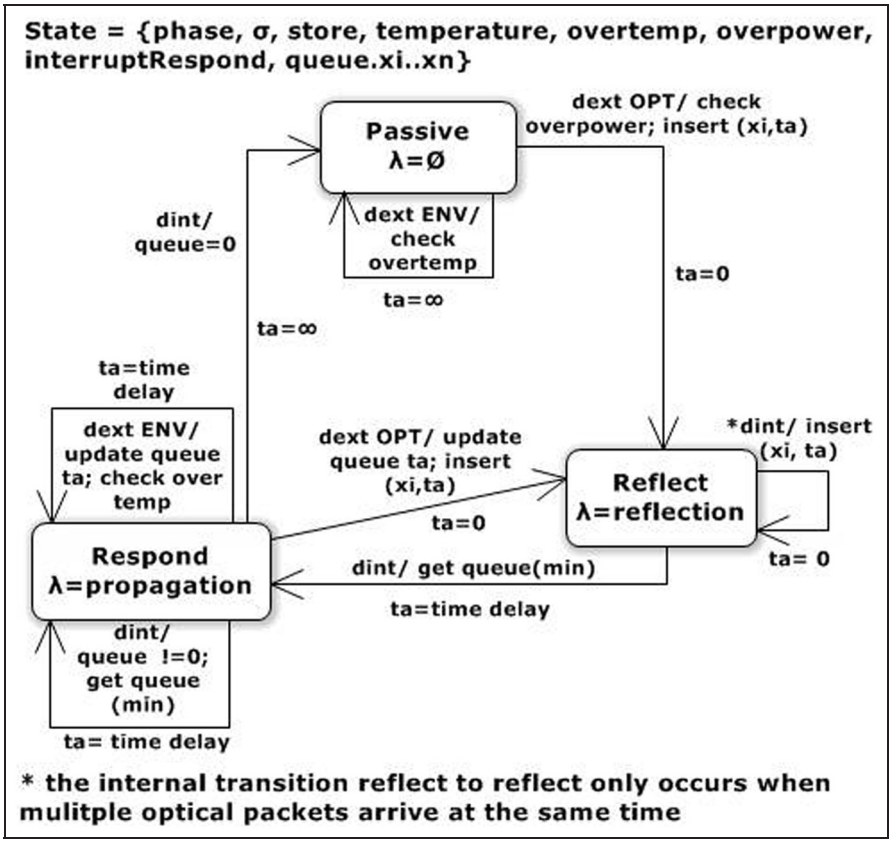

Figure 12. Isolator Discrete Event System Specification (DEVS) phase transition diagram.

The phase transition diagram shows a Passive phase, where the component awaits input. This phase has a time advance (*ta*) of infinity, meaning it will never reach the point of having an output or internal transition. Once an external event occurs, either an optical packet or an environmental message arriving, the device responds through an external transition. A self-loop labeled with ''dext ENV/check overtemp'' (external environmental) represents received environmental messages. The device stores the new temperature and checks against the damaged temperature threshold, setting the *overtemp* state variable to true if reaching it. The internal transition function has output functions *calcForward and calcReverse* that use the current component temperature to determine if the component is in the degraded state. Initially, there was a DEVS degraded state variable, but the research showed it was more efficient to use the output functions to track this state. The time advance for the follow-on phase (e.g., returning back to Passive) is on the other side of the arc; in this case showing ''ta=∞'' as the time advance for the Passive phase is infinity.

If an optical packet arrives, different external transition logic executes, this time checking to see if the arriving optical power exceeds the degraded or damaged thresholds.

Since DEVS collects all messages arriving at the same time into a single message bag as an unordered collection, the external transition from the Passive phase iterates through the bag, checking for the overpower conditions and placing the events in the queue, finally selecting one at random for reflection. Each optical packet enters the queue as event *xi* with some time *ta*, the time advance the packet will remain in the component. The only choices for Passive phase external events are an environmental or optical message; the isolator ignores any other event that arrives.

The transition arc between the Passive and Reflect phases with ''dext OPT/check overpower; insert (xi,ta)'' represents the Reflect phase external optical event and shows the *ta* for the Reflect phase is equal to zero. This phase has two external optical events, one coming from the Passive phase and one from the Respond phase.

In DEVS, the output function λ executes only before an internal transition. The Passive phase does not have an output function, as the *ta* for the phase is infinity, meaning there will never be an internal transition. The ''λ =0'' in

the Passive phase rectangle shows this. The Reflect phase output is the optical packet reflection and the Respond phase output is the propagated optical packet.

The Reflect phase has two possible choices for the internal transition, *dint*, with the note at the bottom specifying the self-loop ''dint/insert(xi,ta)'' executes only when multiple optical packets arrive at the same time. If more packets need reflecting, the self-loop internal transition is called. The Reflect phase checks the queue for any packet not yet reflected, selects it and emits it. As noted before, the time advance of this phase is zero, so this takes place in instant simulation time until complete. This follows the optical SME guidance that reflections must take place before any other operation.

Once done with all reflections, the Reflect phase enters the other internal transition, which removes the queued packet with the shortest time left in its time delay and sends it to the Respond phase. The transition between Reflect and Respond phases labeled, ''dint/get queue(min),'' gets the minimum value in the queue. The time advance shown on the other side of the transition arc is ''time delay,'' indicating a variable time for the selected packet; this becomes the time advance for the Respond phase.

The Respond phase holds the packet for the time delay, usually equal to propagation delay of the device (the time it takes for the optical packet to enter one side and emit out the other). In this phase, we see both internal and external transitions. Since the time advance of the phase is not equal to zero, it is susceptible to external event interruption. As the isolator responds to both environmental and optical external events, it needs transitions for both. For the environmental external event, it is the same for the Passive phase: a check and return to the point of interruption shown by ''dext ENV/update queue ta; check overtemp.'' The difference here is the time spent in the phase, equal to *e*, subtracts from every packet in the queue and the optical packet transitioning the phase. This is shown by the ''update queue ta'' on the *dext* ENV transition arc.

If an optical packet arrives, the ''dext OPT/update queue ta; insert(xi,ta)'' transition arc leaves the Respond phase to the Reflect phase, as it follows the rule all reflections happen before anything else. Here we note the ''update queue ta'' happens again and the queue stores the new packets with the ''insert(xi,ta).'' The component performs the Reflect phase for the new packets, then returns to the Respond phase.

But what happened to the packet that was in the Respond phase? This is where we use the *interruptRespond* state variable. This sets to true for an interrupted phase so the logic in the Reflect phase knows to return to the Respond phase without removing a new packet from the queue, with a new time advance equal to the time remaining for that interrupted packet. Back in the Respond phase, the packet remains for the remainder of the time delay, then the output function propagates the packet, as indicated by the ''λ= propagation'' notation under the name of the phase. The internal *dint* function has two choices: if the queue does not equal zero, it draws the minimum-time packet out of the queue for propagation; if the queue is empty, it advances to the Passive phase to await the next event.

Interrupted packets present a design difficulty, as each component changes a propagating packet during the output function, rather than during input. This decision not only ensures state changes from an environmental or control event affect the propagating packet, but also allows light entering the component behind the packet to affect it. The change to one packet is a minor effect when compared to the large number of packets that travel through components and the simulation, of the order of 1×10^8 packets per second. This design decision came from a discussion between the modeler, an optical physics SME, and the end user. Here we see the simulation design process in action where all parties involved agree to the model choices. Getting agreement on the necessary accuracy of the model is a way to increase the validity of the simulation.

The timescale chosen for the model allows for several design choices. Using the picosecond as the base time allows for discrete events to model continuous time events, as mentioned earlier. This small period means that components only affect a few optical packets during each time unit (typical propagation time for a component is five picoseconds). For example, we chose to change the temperature of a device instantly. This is not true to the real system but the team decided this fast temperature change would affect relatively few packets and simplifies the design of the component. Once again, the users, modelers, and optical SME accepted and agreed on this design abstraction. The timescale allows for the assumption that only one control and environmental message arrives to these ports at any given time, again simplifying the design. The Appendix contains the complete DEVS pseudocode for the isolator.

5.2.2 Isolator reference architecture design. The architecture design of the isolator is a two port, bidirectional optical component used to transmit optical pulses in the forward direction and highly attenuate optical pulses passing in the reverse direction, known as isolated pulses. In QKD systems, isolators can be used to prevent light from reflecting into a laser light source or reflecting out of detectors.

Each optical pulse has the following parameters within its model, as shown in Table 2.

The isolator has two parameters that define its effects on the optical pulse, the isolation loss and the insertion loss, as shown in Table 3.

Table 2. Coherent pulse optical model parameters.

Variable	Model parameter name	Units	Data type	Legal values
Type	*PulseType*	N/A	*Enum*	*CW or Gaussian*
Duration	*Duration*	*seconds*	*Double*	*Real, positive*
E_0	*Amplitude*	$V/meter$	*Double*	*Real*
ω	*CentralFrequency*	$radians/second$	*Double*	*Real, non-negative*
θ	*GlobalPhase*	*radians*	*Double*	*Real*
α	*Polarization*	*radians*	*Double*	*Real*
ϕ	*Ellipticity*	*radians*	*Double*	*Real*
n	*NumberOfGaussians*	N/A	*Int*	*Positive, integer*
A_n	*GaussianAmplitude*	N/A	*Double*	*Real*
μ_n	*GaussianMean*	*seconds*	*Double*	*Real, non-negative*
σ_n	*GaussianStandardDeviation*	*seconds*	*Double*	*Real, non-negative*
$\Delta\omega$	*DeltaCentralFrequency*	$radians/second$	*Double*	*Real, non-negative*

Table 3. Isolator.

Parameter	Description	Data type	Units	Minimum	Maximum	Default
Isolation loss	The magnitude of attenuation applied to pulses travelling in the reverse, unintended direction	Double	Decibels (dB)	0.0	80.0	50.0
Insertion loss	The magnitude of insertion attenuation	Double	Decibels (dB)	0.0	80.0	0.1

The amplitude of the isolated pulse is calculated using the amplitude of the input pulse, insertion loss, and isolation loss, as shown in Equation (1):

$$Amplitude_{Isolated} = Amplitude_{Input} * \sqrt{10^{\frac{-(InsertionLoss + IsolationLoss)}{10}}} \quad (1)$$

The amplitude of the output pulses is calculated just using the amplitude of the input pulse and insertion loss, as shown in Equation (2):

$$Amplitude_{Output} = Amplitude_{Input} * \sqrt{10^{\frac{-InsertionLoss}{10}}} \quad (2)$$

The modeled behaviors are based on Thor Labs[48] and Chang and Sorin[49] and design input from the optical physics SME.

5.3 DEVS model of the classical pulse generator

Since discussing the isolator design in depth, we can look at the rest of the CPG components in comparison. The isolator design follows the base design for all optical components with a Passive phase, a Reflect phase, and some form of a Respond phase. The isolator, polarizer, bandpass filter, and the beamsplitter all share this common design. The beamsplitter differs by having four optical ports rather than two and splits an incoming optical into two propagating packets.

The laser and classical detector differ by having electric circuits in the device. They have an electric control port for control messages and a fourth phase to update the control logic within the device. The laser is unique as the only device to create optical pulses and the classical detector receives optical pulses and outputs a control message to the coupled controller. Response to optical pulses and environmental messages is the same in these two devices.

The PM fiber is a simple device with Passive and Respond phases because of the design decision that fiber does not create reflections when receiving optical pulses. It responds in the same manner as the isolator to environmental messages and its Respond phase works the same. The largest difference is the fiber deletes optical pulses if their power is below a specified limit.

The DEVS CPG model is a coupled model meaning is it comprised of atomic models. Figure 13 shows the boundary of the CPG and the components. The CPG has environmental and control input and output ports on the left and optical input and output ports on the right. The CPG model has no functions or phases like other DEVS models, but because of the DEVS's composability and closure properties, it behaves as an atomic model. The inputs from external CPG ports connect to input ports on internal components and the output from one or more internal components connects to the CPG output ports. Internal components connect through component input and output ports.

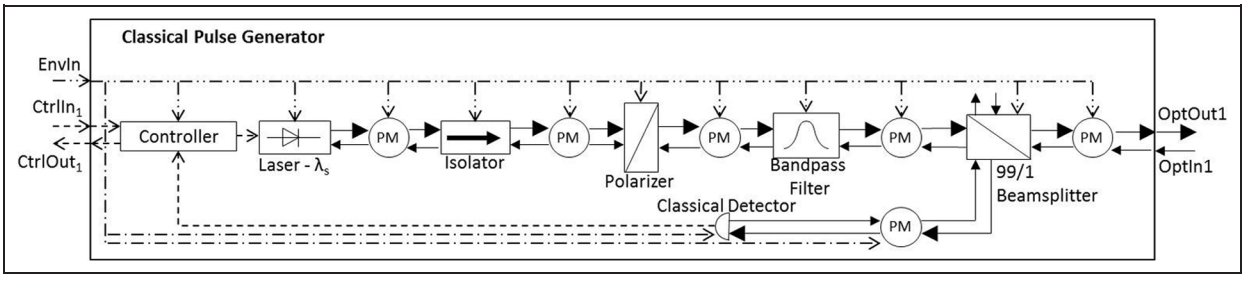

Figure 13. Conceptual Discrete Event System Specification (DEVS) architecture of the classical pulse generator (CPG).

Table 4. Messages received by the classical pulse generator (CPG) controller from the quantum module controller.

Input messages	Response
CPG_ENV	Set the internal CPG controller temperature
CPG_RESET	Resets the CPG controller and clears variables
CPG_STATUS_ REQUEST	Sends the CPG controller status and stored magnitude value
CPG_FIRE_ LASER	Sends a "Fire" command to the laser

Table 5. Messages received by the classical pulse generator (CPG) controller from the classical detector.

Input messages	Response
CD_DETECTION	Store the magnitude of the detected laser pulse

Table 6. Messages sent from the classical pulse generator (CPG) controller to the quantum controller.

Output messages	Content
CPG_ACK	Response to a Reset message
CPG_STATUS	Response to a Status Request message

Table 7. Messages sent from the classical pulse generator (CPG) controller to the laser.

Output messages	Content
CPG_LASER_ FIRE	Command to fire the laser one time

The optical path starts with the laser and ends with the PM fiber linked to the CPG optical output. The primary travel direction for optical packets is indicated by the larger arrowhead in Figure 13. Each of the optical components has a bidirectional optical connection shown by the arrows between each component. This is because the DEVS formalism decomposes a bidirectional connection into two separate unidirectional connections. The PM fiber connected to the beamsplitter output sends the optical packet to the CPG external port, which sends the packet to the next subsystem. Each component has a one-way environmental port shown by the dotted-dashed lines. These ports connect to higher functions that send temperature updates to each component. Finally, there are connections from the CPG external and internal control ports to the controller and control port connections between the controller and laser and classical detector and controller.

The CPG controller is a simple abstraction of the electric circuits connecting devices in the CPG to the quantum controller. Its basic functions include receiving control messages from higher functions, passing messages to the laser and responding to status requests from higher functions. It receives the output control messages from the classical detector and stores detection data. It has DEVS pseudocode and functions as an atomic model.

Table 4 lists the messages from the quantum module controller to the CPG controller, Table 5 lists the messages from the classical detector to the controller, Table 6 lists the messages sent by the CPG controller to the quantum module controller, and Table 7 lists the messages from the controller to the laser.

The DEVS model of the CPG does not have the external, internal, output, confluence, and time advance functions like an atomic model. Instead, the formalism specifies a set of all inputs, a set of all outputs, a list of internal model names, a list of the atomic models that comprise the coupled model, a list of external input and output connections, and a list of internal component input and output connections. The Appendix contains the DEVS pseudocode for the CPG and its controller.

6. Discussion

6.1 Discoveries made during the modeling process

Discoveries made during the conceptual modeling process fall into two categories: the modeling relation and the

simulation relation. Recall the modeling relation is a measure of how close the model is to the source system and the simulation relation is a measure of how well the simulation executes the conceptual model instructions.

6.1.1 The modeling relation. The original DEVS model failed to consider the change in attenuation in the electronically variable optical attenuator (EVOA) happens over a time period determined by the device rate-of-change and the difference between the new and old attenuation values. Discussion with the optical SME uncovered this difference and lead to changing the DEVS model to an accuracy acceptable to the research team. The problem with the EVOA identified the same condition with the polarization controller. In this case, it highlighted not only a necessary change in the components, but a change in the mathematical models for these components, as the necessary parameters were not in the original models.

Both of these rate-of-change problems necessitated a change to the MS4ME model that was not in line with pure DEVS theory, due to the current design of MS4ME. This simulator uses external, output, and internal events with associated tagged blocks of code to handle these transitions. In the version used for this research, a call for the time advance in the internal event returns the time advance of the *next* phase, rather than the time advance of the current phase. This made calculating for the rate-of-change difficult, as the time spent in the current phase was necessary. After consultation with the MS4 Systems modeling team, including Dr Zeigler, we found an acceptable fix within MS4ME for the rate-of-change issue by adding a local variable to the output event to hold the time spent in the current phase.

The DEVS modeler noticed the current QKD demonstration architecture uses only the polarization-independent version of the isolator, but other architectures use the polarization-dependent version. The optical SME realized the need to provide the mathematical model for the polarization-dependent version of the device for the DEVS model. The updated math model permitted the DEVS modeler to update the conceptual model to apply to both forms of the isolator.

6.1.2 The simulation relation. Identifying the change to the EVOA DEVS model made the software modelers aware that these components in the proof-of-concept simulation had the same ''instantaneous change'' flaw. Using the updated conceptual model and input from the SME allowed for updates to the simulation code to capture the proper component rate-of-change.

Early in the DEVS modeling of pulsed light, we recognized the proof-of-concept simulation needed significant changes to handle continuous-wave light. The simulator models short-duration pulses using picosecond scales, but continuous-wave light cannot use the same abstraction method, requiring changes to the QKD simulator.

In the DEVS model, changes to the component during each optical packet propagation time affect the packet as it propagates through the component. Conversely, the QKD simulation receives a packet and schedules it for a future propagation. Since the component could change during the time before propagation, the scheduled optical pulse may not display proper behavior. This may require a change to how the QKD simulator handles optical packets. This speaks to the composability of DEVS and having well-defined behavior in the models.

6.2 Did the research increase the validity of qkdX?

The concept of validity is not one of percentages or finite measurements. As defined earlier in this paper, validity is an expression of how difficult it is to differentiate between the model and source system outputs for the given experimental frame. Sargent suggests that ''acceptance'' of the model's accuracy for its intended purpose is the measure of conceptual model validity. Further he states that each submodel and the overall models need evaluation to determine if they are correct for the purpose of the model.[21] Two of the primary techniques for this are *face validation* and *traces*.

Face evaluation is where experts in the problem area evaluate the conceptual model to determine if it is correct and reasonable, usually by examining flowcharts or graphical models or a set of model equations.[21] In this research, we have research partners who are experts in quantum mechanics, QKD, physics, and optics that constantly review the optical models and provide feedback and corrections as necessary.

Traces involve tracking entities through each atomic and coupled model determining if the logic and behavior associated with each is proper while maintaining necessary accuracy throughout.[21] The MS4ME simulator provided visual representations of the components as they transited through the models and produced detailed output to check the accuracy of the models during these tests.

As the models developed, the SMEs and research team provided feedback for correction, drawing each model closer to the expected system behavior, until each was deemed ''acceptable,'' as discussed in Section 3.1.2. Using the definitions of validity discussed earlier, this effort provided models that captured the required behavior and met the required accuracy, and so are considered ''valid'' with the understanding this validity only applies to the models built for the specific experimental frame. Any change to the experimental frame lessens or negates the validity of the models.

6.3 Benefits of using DEVS

6.3.1 Mathematically proven formalism for creating a conceptual model. With the understanding that a DES is a finite state machine with a set of triples of inputs, states, and outputs to describe each state, DEVS captures these in a logical manner and provides a formal language to describe the conceptual model. DEVS uses set theory to prove its applicability to DES models.[50]

6.3.2 Tool-independent form of the model. One of the central ideas of DEVS is the model is the bridge between the source system and the simulation. There is no direct connection between the source system and simulation, meaning the conceptual model created using DEVS is applicable to any simulation, if the simulation is capable of implementing the DEVS-specified behavior. A DEVS conceptual model is independent of a specific simulation.

Some simulation environments have their own programming language while others use standard programming languages (i.e., OMNeT++ uses the C++ language). Each language may have idiosyncrasies that prevent easy translation from one simulation environment to another.[51] For an example, the Java language (used in MS4ME) has routines to free memory storage (garbage collection) but C++ does not; the Java runtime system has ways of knowing the size of an array, but the C++ runtime does not.[51–53] The resulting models are based solely on the language rules inherent in DEVS, and not on the selected simulation environment. This allows for creation of models usable for any simulation environment. The modeler needs only to express the DEVS behavior in the simulator of choice.

6.3.3 Closure under coupling within the formalism. If any result from coupling components specified by the formalism can itself be specified by the formalism, then it has the *closure under coupling* property.[54] This property, also called composability, ensures DEVS models connect in any manner, produce expected behavior, and the coupled models are behaviorally equivalent to atomic models. Together with tool-independence, these properties allow for using repositories of models in a ''mix and match'' environment, allowing the modeler to build hierarchal systems from smaller subsystems and components.

6.3.4 Canonical understanding of the model behavior. DEVS forces the modeler to carefully consider all facets of desired behavior within the model, including all inputs, outputs, and timing segments. This greatly reduces or eliminates emergent or unexpected behavior. By considering ''real-time'' transitions, well-defined behavior, and tool-independence, the modeler creates a deep understating of the system usable in any modeling simulation. By understanding the canonical behavior of the model, the modeler can implement changes to decrease the differences between the model and the referent system, thereby increasing the validity of the model for the chosen experimental frame. By delving into the source system and creating a set of atomic models not further decomposable, the modeler creates a deep understanding of the system usable in any modeling simulation.

6.3.5 Well-defined behavior in the conceptual model. Having a canonical understanding enables the modeler to create a set of rules for each atomic and coupled model, creating well-defined behaviors. A concern for complex simulations is ill-defined or emergent behavior that leads to the simulation entering a state or condition it cannot leave. Simple simulations can be checked by evaluating every possible state, but with a large simulation, the state-space becomes unmanageable and infeasible to check. Project time and funds limit testing for simulations, even with automated support.

Using DEVS for modeling prevents these types of problem behaviors by requiring the modeler to identify the lowest level of objects in the simulation, one that cannot be further decomposed. These atomic models can be modeled regardless of their immediate context (i.e., the component is isolated from any outside influences) and their total behavior specified. A correctly specified model operates properly regardless of its use and the environment. DEVS defines the conditions necessary to ensure this well-defined behavior.

6.4 Limitations of using DEVS

6.4.1 Applying DEVS to complex components. Much of the written material available for DEVS in textbooks and papers provides only simplistic examples. Even the more complex examples available through RTSync provided little guidance to model the optical components. This complexity led this researcher to contact the ACIMS for advice on using DEVS for our unique, innovative models. He provided suggestions on how to use DEVS to model the timing issues at the picosecond scale and capturing wave and particle behavior.

It can be difficult to verify the DEVS pseudocode to the source system behavior, especially true when modeling predicted or notional systems. Since our demonstration QKD system is built from real optical components but in a notional architecture, we had the difficulty of verifying the component behavior. Our solution to this problem greatly increased the DEVS work by necessitating programming the pseudocode twice, once into MS4ME, and then into the selected *qkdX* simulator.

6.4.2 DEVS and processes and information flows. DES models are increasing in complexity and being used in new information-centered fields. DEVS was not created for these types of problems and has difficulty expressing processes and information flows within the formalism. Heretofore, the solution has been to create subsets of DEVS to handle the specific problem. This is leading to a fragmentation of the formalism and makes it hard to determine which form of DEVS is appropriate for the modeling problem.

6.4.3 Not visual without using a DEVS simulator. DEVS is a set of language rules to formally describe a problem. There is no visual component to DEVS unless the modeler uses a DEVS-compliant simulation program. While very useful for being tool-independent, this requires the modeler to use that particular simulator's functions, which may not necessarily conform completely to DEVS, as seen with the EVOA timing issue and MS4ME.

7. Conclusions

In this paper, we demonstrated how the DEVS formalism could increase the validity of a QKD simulation for its designed purpose. The question of what is sufficiently accurate is the question of validity, as noted by Zeigler and Robinson. When used properly, DEVS increases the validity of simulation, for its intended purpose. No simulation is valid for all purposes, so it is important to understand Zeigler's idea of the experimental frame so to carefully limit discussing validity to an intended purpose.

To test our hypothesis, we created DEVS models for atomic components in the Alice CPG and programmed them into a DEVS-compliant simulator to check their correctness. The SMEs and the research team reviewed the results (face validity) and checked the output values against the required accuracies as events moved through the models (tracing). After correction, the atomic models were used in a coupled model, the CPG, demonstrating the hierarchal properties of DEVS.

Using DEVS allowed the team to refine the *qkdX* simulation, correcting several errors and aiding the research team in recognizing missing behaviors within the simulation. DEVS increased the validity of the *qkdX* optical pathway by aiding the team in making *qkdX* simulation behavior closer to the source system behavior, showing DEVS can be used to increase validity by creating optical component models fit for the purposes of the simulation and acceptable to the community of developers and users.

Disclaimer

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the US Government.

Funding

This work was supported by the Laboratory for Telecommunication Sciences (grant number 5713400-301-6448).

8. References

1. Singh S. *The code book: the secret history of codes and code-breaking*. London: Fourth Estate, 1999, p.ix.
2. Barker EB, Barker WC and Lee A. Guideline for implementing cryptography in the federal government, NIST, Gaithersburg, MD, http://csrc.nist.gov/publications/nistpubs/800–21–1/sp800–21–1_Dec2005.pdf (2005, accessed 3 February 2014).
3. Shannon CE. Communication theory of secrecy systems, Bell Labs. USA, http://dm.ing.unibs.it/giuzzi/corsi/Support/papers-cryptography/Communication_Theory_of_Secrecy_Systems.pdf (1946, accessed 3 February 2014).
4. Shannon CE. A mathematical theory of communication. *The Bell System Technical Journal 27*, pp. 379–423, http://www.enseignement.polytechnique.fr/profs/informatique/Nicolas.Sendrier/X02/TI/shannon.pdf (1948, accessed 29 September 2014).
5. Schneier B. *Applied cryptography: protocols, algorithms, and source code in C*. 2nd ed.New York: John Wiley & Sons, 1995, pp.151–153.
6. Grimaila MR, Morris JD and Hodson D. Quantum Key Distribution, a revolutionary security technology. *ISSA J* 2012; 27: 20–27.
7. ID Quantique SA. Redefining security Geneva government secure data transfer for elections, http://www.idquantique.com/images/stories/PDF/cerberis-encryptor/user-case-gva-gov.pdf (2011, accessed 3 February 2014).
8. Morris JD, Grimaila MR, Hodson DD, et al. A survey of Quantum Key Distribution (QKD) technologies. In: Akhgar B and Arabnia HR (eds) *Emerging trends in ICT security*. 1st ed. Waltham, MA: Elsevier, 2013, pp.141–152.
9. Austrian Institute of Technology. QKD Software, https://sqt.ait.ac.at/software/projects/qkd-software (2014, accessed 3 February 2014).
10. Morris JD, Hodson DD, Grimaila MR, et al. Towards the modeling and simulation of quantum key distribution systems. *International Journal of Emerging Technology and Advanced Engineering* 4(2), http://www.ijetae.com/files/Volume4Issue2/IJETAE_0214_143.pdf (2014, accessed 6 January 2014).
11. Zeigler BP, Ball G, Cho H, et al. Implementation of the DEVS formalism over the HLA/RTI: problems and solutions. In: *the simulation interoperation workshop*, http://acims.asu.edu/wp-content/uploads/2012/02/SIWDEVSImplemHLARTI.pdf (1999, accessed 3 February 2014).
12. Zeigler BP, Song HS, Kim TG, et al. DEVS framework for modelling, simulation, analysis, and design of hybrid systems, http://www.researchgate.net/publication/2684908_DEVS_Framework_for_Modelling_Simulation_Analysis_and_

Design_of_Hybrid_Systems/file/72e7e5215bfca6eaeb.pdf (1995, accessed 3 February 2014).
13. Wainer GA and Giambiasi N. Application of the cell-DEVS paradigm for cell spaces modelling and simulation. *Simulation* 2001; 76: 22–39.
14. Zeigler BP, Kim D and Buckley SJ. Distributed supply chain simulation in a DEVS/CORBA execution environment. In: *31st winter simulation conference*, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.9681rep=rep1type=pdf (1999, accessed 3 February 2014).
15. Mittal S, Risco JL and Zeigler BP. DEVS-based simulation web services for net-centric T&E. In: *the 2007 summer computer simulation conference*, http://cell-devs.sce.carleton.ca/citations/SC208_Mittal.pdf (2007, accessed 3 February 2014).
16. Ntaimo L, Zeigler BP, Vasconcelos MJ, et al. Forest fire spread and suppression in DEVS. *Simulation* 2004; 80: 479–500.
17. Wainer G. Applying cell-DEVS methodology for modeling the environment. *Simulation* 2006; 82: 635–660.
18. Gunay HB, O'Brien L, Goldstein R, et al. Development of discrete event system specification (DEVS) building performance models for building energy design. In: *the symposium on simulation for architecture & urban design*, http://www.autodeskresearch.com/pdf/46_Final_Paper.pdf (2013, accessed 3 February 2014).
19. Balci O. Principles and techniques of simulation validation, verification, and testing. In: *the 1995 winter simulation conference*, http://www.informs-sim.org/wsc95papers/1995_0021.pdf (1995, accessed 3 February 2014).
20. Balci O. Verification validation and accreditation of simulation models. In: *the 29th winter simulation conference*, http://www.informs-sim.org/wsc97papers/0135.PDF (1997, accessed 3 February 2014).
21. Sargent RG. Verification and validation of simulation models. In: *presented at the 37th winter simulation conference*, http://student.telum.ru/images/6/66/Sargent_VV_2010.pdf (2005, accessed 3 February 2014).
22. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed.San Diego, CA: Academic Press, 2000, pp.75–76.
23. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed.San Diego, CA: Academic Press, 2000, p.142.
24. Zeigler BP and Sarjoughian H. *Guide to modeling and simulation of systems of systems*. London: Springer, 2013, p.24.
25. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed.San Diego, CA: Academic Press, 2000, pp.149–152.
26. Balci O, Arthur JD and Ormsby WF. Achieving reusability and composability with a simulation conceptual model. *J Simulat* 2011; 5: 157–165.
27. Hofmann M, Palii J and Mihelcic G. Epistemic and normative aspects of ontologies in modelling and simulation. *J Simulat* 2011; 5: 135–146.
28. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed.San Diego, CA: Academic Press, 2000, pp.25–26.
29. Tolk A, Diallo SY, Padilla JJ, et al. Reference modelling in support of M&S—foundations and applications. *J Simulat* 2013; 7: 69–82.
30. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed.San Diego, CA: Academic Press, 2000, p.76.
31. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed.San Diego, CA: Academic Press, 2000, pp.152–155.
32. Zeigler BP and Sarjoughian HS. Approach and techniques for building component-based simulation models, http://acims.asu.edu/wp-content/uploads/2012/02/iitsec.ppt (2005, accessed 3 February 2014).
33. Robinson S. The future's bright the future's… conceptual modelling for simulation!*J Simulat* 2007; 1: 149–152.
34. Robinson S. *Simulation: the practice of model development and use*. Chichester: John Wiley & Sons, 2004, p.65.
35. Robinson S. *Simulation: the practice of model development and use*. Chichester: John Wiley & Sons, 2004, pp.77–78.
36. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed.San Diego, CA: Academic Press, 2000, p.31.
37. Szabo C and Teo YM. An analysis of the cost of validating semantic composability. *J Simulat* 2012; 6: 152–163.
38. Vargas A. OMNeT++. In: *Modeling and tools for network simulation*. Wehrle, K., Günes, M., Gross, J.(Eds.) Berlin: Springer, 2010, pp.35–59.
39. Sung C and Kim TG. Collaborative modeling process for development of domain-specific discrete event simulation systems. *IEEE Trans Syst Man Cybern Part C Appl Rev* 2012; 42: 532–546.
40. Wolfram. Wolfram Mathematica, http://www.wolfram.com/mathematica/ (2014, , accessed 3 February 2014).
41. MS4 Systems. MS4ME, http://www.ms4systems.com/pages/ms4me.php (2014, accessed 3 February 2014).
42. Arizona State University. Arizona Center for Integrative Modeling and Simulation, http://acims.asu.edu/ (2014, accessed 3 February 2014).
43. Zeigler BP, Sarjoughian HS and Au W. Object-oriented DEVS: object behavior specification. In: *proceedings of enabling technology for simulation science*, Orlando, FL, 1997, Vol. 3083.
44. Traoré MK and Muzy A. Capturing the dual relationship between simulation models and their context. *Simulat Model Pract Theor* 2006; 14: 126–142.
45. Bennett CH and Brassard G. Quantum cryptography: public key distribution and coin tossing. In: *the IEEE international conference on computers, systems and signal processing*, http://www.cs.ucsb.edu/~chong/290N-W06/BB84.pdf (1984, accessed 3 February 2014).

46. ID Quantique. Quantus Optical Random Number Generator, http://www.idquantique.com/component/content/article.html?id=9 (2014, accessed 3 February 2014).
47. Zeigler BP, Praehofer H and Kim TG. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed.San Diego, CA: Academic Press, 2000, pp.89–93.
48. Thor Labs. IR Fiber optic isolators with SM fiber (1290 - 2010 nm), http://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=6178 (2014, accessed 3 February 2014).
49. Chang KW and Sorin WV. High-performance single-mode fiber polarization-independent isolators. *Opt Lett* 1990; 15: 449–451.
50. Zeigler BP. *Theory of modeling and simulation*. New York: John Wiley & Sons, 1976, pp.293–301.
51. Miller JR. Moving from java to c++, http://people.eecs.ku.edu/~miller/Courses/JavaToC++/JavaToC++.html (2013, accessed 3 February 2014).
52. Miller JR. Moving from java to c++: arrays, http://people.eecs.ku.edu/~miller/Courses/JavaToC++/Arrays.html (2013, accessed 3 February 2014).
53. Miller JR. Moving from java to c++: memory management, http://people.eecs.ku.edu/~miller/Courses/JavaToC++/MemoryManagement.html (2013, accessed 3 February 2014).
54. Barros FJ. Modeling formalisms for dynamic structure systems. *ACM Trans Model Comput Simulat* 1997; 7: 501–515.

Author biographies

Jeffrey D Morris (BS 2001, MMIS 2006, MSSI 2007, PhD 2014, Air Force Institute of Technology) is a Master Sergeant in the US Army and currently serves as a researcher and instructor at the Army Cyber Institute, US Military Academy, West Point, New York, USA. He can be contacted at jeffrey.d.morris18.mil@mail.mil.

Michael R Grimaila, PhD, CISM, CISSP (BS 1993, MS 1995, PhD 1999, Texas A&M University) is a Professor of Systems Engineering and member of the Center for Cyberspace Research (CCR) at the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio, USA. He is a member of the ACM, a Senior Member of the IEEE, and a Fellow of the ISSA. His research interests include computer engineering, mission assurance, quantum communications and cryptography, data analytics, network management and security, and systems engineering. He can be contacted via email at Michael.Grimaila@us.af.mil.

Douglas D Hodson, PhD (BS 1985, MS 1987, PhD 2009, Air Force Institute of Technology) is an Assistant Professor of Software Engineering at the AFIT, Wright-Patterson AFB, Ohio, USA. His research interests include computer engineering, software engineering, real-time distributed simulation, and quantum communications. He is also a DAGSI scholar and a member of Tau Beta Pi. He can be contacted at Douglas.Hodson@us.af.mil.

Colin V McLaughlin, PhD (BA 2003, PhD 2010, University of Maryland, Baltimore County) is a Research Physicist at the United States Naval Research Laboratory, Washington, D.C., USA. He specializes in photonic communication devices and systems. He can be contacted at Colin.McLaughlin@nrl.navy.mil.

David Jacques, PhD (BS 1983, MS 1989, PhD 1995 Air Force Institute of Technology) is an Associate Professor of Systems Engineering at the AFIT, Wright-Patterson AFB, Ohio, USA. His research interests are in the areas of concept definition and evaluation, architecture modeling, and optimal system design. Dr Jacques had prior military assignments in intelligence analysis, research and development, and test and evaluation. He can be contacted at David.Jacques@us.af.mil.

Appendix

A Isolator DEVS pseudocode

$DEVS_{Isolator} = (X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$

External Transition Function:

δ_{ext}(*phase, σ, store, temperature, overtemp, overpower, interruptRespond, queue*) =
(“reflect”, 0, *store, temperature, overtemp, overpower,interruptRespond, queue.x*1..*xn*)
  if *phase* = “passive” and *p* ∈ {“$OptIn_1$”, “$OptIn_2$”}
    for *messagebag* != null
      current = messagebag_first()
      if current.value.power > *damaged.power*
      overpower = “Y”
  insert_event_q(*current*)
  remove_event_m(*current*)
queue.current = queue_first(*queue*)
reflect = (*queue.current.p*), calcReflected(*queue.current.v*))
mark_reflected(*queue.current*)
interruptRespond = “N”

(“reflect”, 0, *store, temperature, overtemp, overpower, interruptRespond, queue.x*1..*xn*)
  if *phase* = “respond” and *p* ∈ {“$OptIn_1$”, “$OptIn_2$”}
    update_delay(*queue*)
    for *messagebag* != null
      current = messagebag_first()
      if current.value.power > *damaged.power*
      overpower = “Y”
    insert_event_q(*current*)
     remove_event_m(*current*)
  queue.current = queue_need_reflected()
  reflect = (*queue.current.p*), calcReflected(*queue.current.v*))
    mark_reflected(*queue.current*)
    interruptRespond= “Y”
    timeLeftRespond = *timeLeftRespond* - *e*

(“passive”, ∞, *store, temperature, overtemp, overpower, interruptRespond, queue.x*1..*xn*)
  if *phase* = “passive” and *p* = “EnvIn”
  temperature = *messagebag.temperature*
  if *temperature* > *damage.temp*
    overtemp = “Y”
(“respond”, *time.delay, store, temperature, overtemp, overpower, interruptRespond, queue.x*1..*xn*)
  if *phase* = “respond” and *p* = “EnvIn”
    update_delay(*queue*)
    timeLeftRespond = *time.delay- e*
    temperature = *messagebag.temperature*
    if *temperature* > *damage.temp*
      overtemp = “Y”
    time.delay = *timeLeftRespond*

(*phase, σ– e, store, temperature, overtemp, overpower, interruptRespond, queue.x*1..*xn*)
otherwise;

Internal Transition Function:

δ_{int}(*phase, σ, store, temperature, overtemp, overpower, interruptRespond, queue, e*, ((p_i,v_i),…. (p_n,v_n))) =
(“reflect”, 0, *temperature, overtemp, overpower, interruptRespond, queue.x*1..*xn*))

if *phase* = ‘‘reflect’’ and *need.reflect* != null
need.reflect = queue_need_reflected()
current = *need.reflect*
reflect = (*current.p*), calcReflected(*current.v*))
mark_reflected(*current*)

(‘‘respond’’, *time.delay, store, temperature, overtemp, overpower, interruptRespond, queue.x*1..*xn*)
if *phase* = ‘‘reflect’’ and *need.reflect* = null
need.reflect = queue_need_reflected()
if *interruptRespond* = ‘‘N’’
current = queue_min()
time.delay = current.time.delay
if InPort = ‘‘$OptIn_1$’’
outputPulse = calcForward(*current.v, temperature, overtemp, peakpwr, overpwr*)
outputPort =‘‘$OptOut_2$’’
if InPort = ‘‘$OptIn_2$’’
outputPulse = calcReverse(*current.v, temperature, overtemp, peakpwr, overpwr*)
outputPort =‘‘$OptOut_1$’’
timeLeftRespond = propagation delay
else
time.delay = *timeLeftRespond*

(‘‘respond’’, *time.delay, store, temperature, overtemp, overpower, interruptRespond, queue.x*1..*xn*)
if *phase* = ‘‘respond’’ and *size* > 0
update_delay(*queue*)
size= queue_size()
current = queue_min()
time.delay = current.time.delay
if InPort = ‘‘$OptIn_1$’’
outputPulse = calcForward(*current.v, temperature, overtemp, peakpwr, overpwr*)
outputPort =‘‘$OptOut_2$’’
if InPort = ‘‘$OptIn_2$’’
outputPulse = calcReverse(*current.v, temperature, overtemp, peakpwr, overpwr*)
outputPort =‘‘$OptOut_1$’’
interruptRespond= ‘‘N’’

(‘‘passive’’, ∞, *store, temperature, overtemp, overpower, interruptRespond, queue.x*1..*xn*)
if *phase* = ‘‘respond’’ and *size* = 0
size= queue_size()

Confluence Function:

$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$;

Output Function:

λ(*phase*, σ, *store, temperature, overtemp, overpower*) =
(*reflect.p, reflect.v*)
if phase = ‘‘reflect’’
(*outputPort, outputPulse*)
if phase = ‘‘propagate’’
$\varnothing$ (null output)
otherwise;

Time advance Function:

ta(*phase*, σ, *store, temperature, overtemp, overpower, interruptRespond, queue*) = σ;

B. Classical pulse generator controller DEVS pseudocode

$\boldsymbol{DEVS_{CPGcontroller} = (X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)}$

External Transition Function:

$\delta_{ext}(phase, \sigma, store, temperature, overtemp, overpower, lastCDPower, e, ((p_i,v_i),\ldots. (p_n,v_n))) =$
(“respond”, 0, *store, temperature, overtemp, overpower, lastCDPower*)
if *phase* = “passive” and p = “CtrlIn_1”
ctrlOutput = ctrlMsg(*store*)
if *ctrlMsg.status* = “init” or “get status”
outputPort = “CtrlOut_1”
if *ctrlMsg.status* = “fire laser”
outputPort = “CtrlOut_2”

(“passive”, 0, *store, temperature, overtemp, overpower, lastCDPower*)
if *phase* = “passive” and p = “CtrlIn_2”
lastCDPower = *messagebag.magnitude*

(“passive”, ∞, *store, temperature, overtemp, overpower, lastCDPower*)
if *phase* = “passive” and p = “EnvIn”
temperature = *messagebag.temperature*
if *temperature* > *damage.temp*
overtemp = “Y”

(*phase, σ– e, store, temperature, overtemp, overpower, lastCDPower*)
otherwise;

Internal Transition Function:

$\delta_{int}(phase, \sigma, store, temperature, overtemp, overpower, lastCDPower) =$
(“passive”, ∞, *store, temperature, overtemp, overpower, lastCDPower*)
if *phase* = “respond”

Confluence Function:

$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$;

Output Function:

$\lambda(phase, \sigma, store, temperature, overtemp, overpower, lastCDPower) =$
(*output.port, output.pulse*)
if phase = “respond”

(*outputPort, ctrlOutput*)
if phase = “respond”

∅ (null output)
otherwise;

Time advance Function:
$ta(phase, \sigma, store, temperature, overtemp, overpower, lastCDPower) = \sigma$;

C. Classical pulse generator coupled model pseudocode

$\boldsymbol{DEVS_{CPG} = (X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC)}$

InPorts = {“CtrlIn_1”, “CtrlIn_2”, “OptIn_1”, “OptIn_3”, “OptIn_4”, “EnvIn”}
X = {(“CtrlIn_1”, v), (“CtrlIn_2”, v), (“OptIn_1”, v), (“OptIn_3”, v), (“OptIn_4”, v), (“EnvIn”, v) $|v \in V$}

OutPorts = {“$CtrlOut_1$”, “$CtrlOut_2$”, “$OptOut_1$”, “$OptOut_3$”, “$OptOut_4$”}
Y = {(“$CtrlOut_1$”, v), (“$CtrlOut_2$”, v), (“$OptOut_1$”, v), (“$OptOut_3$”, v), (“$OptOut_4$”, v) $|v \in V$}

D = {controller, laser, isolator, polarizer, bandpass, beamsplitter, classicaldetector, PMfiber}
$M_d = M_{controller}, M_{laser}, M_{isolator}, M_{polarizer}, M_{bandpass}, M_{beamsplitter}, M_{classicaldetector}, M_{PMfiber}$

EIC = {((N, “$CtrlIn_1$”),(controller, “$CtrlIn_1$”)), ((N, “EnvIn”),(controller, “EnvIn”)), ((N, “EnvIn”),(laser, “EnvIn”)), ((N, “EnvIn”),(isolator, “EnvIn”)), ((N, “EnvIn”),(polarizer, “EnvIn”)), ((N, “EnvIn”),(bandpass, “EnvIn”)), ((N, “EnvIn”),(beamsplitter, “EnvIn”)), ((N, “EnvIn”),(classicaldetector, “EnvIn”)), ((N, “EnvIn”),(PMfiber, “EnvIn”)), {((N, “$OptIn_1$”),(PMfiber, “$OptIn_2$”))}

EOC = {((PMfiber, “$OptOut_2$”),(N, “$OptOut_1$”)), ((controller, “$CtrlOut_1$”),(N, “$CtrlOut_1$”))}
IC = {((controller, “$CtrlOut_2$”), (laser, “$CtrlIn$”)), ((laser, “$OptOut_1$”),(PMfiber, “$OptIn_1$”)), ((PMfiber, “$OptOut_2$”), (isolator, “$OptIn_1$”)), ((isolator “$OptOut_2$”), (PMfiber, “$OptIn_1$”)), ((PMfiber, “$OptOut_2$”), (polarizer, “$OptIn_1$”)), ((polarizer “$OptOut_2$”), (PMfiber, “$OptIn_1$”)), ((PMfiber, “$OptOut_2$”), (bandpass, “$OptIn_1$”)), ((bandpass “$OptOut_2$”), (PMfiber, “$OptIn_1$”)), ((PMfiber, “$OptOut_2$”), (beamsplitter, “$OptIn_1$”)), ((beamsplitter, “$OptOut_4$”),(PMfiber, “$OptIn_1$”)), ((beamsplitter, “$OptOut_3$”),(PMfiber, “$OptIn_2$”)), ((PMfiber, “$OptOut_1$”),(classicaldetector, “$OptIn_1$”)), ((classicaldetector, “CtrlOut”),(controller, “$CtrlIn_2$”)), ((classicaldetector, “$OptOut_1$”), (PMfiber, “$OptIn_1$”)), ((PMfiber, “$OptOut_2$”), (beamsplitter, “$OptIn_3$”)), ((PMfiber, “$OptOut_1$”), (beamsplitter, “$OptIn_4$”)), ((beamsplitter, “$OptOut_1$”),(PMfiber, “$OptIn_2$”)), ((PMfiber, “$OptOut_1$”), (bandpass, “$OptIn_2$”)), ((bandpass, “$OptOut_1$”),(PMfiber, “$OptIn_2$”)), ((PMfiber, “$OptOut_1$”), (polarizer, “$OptIn_2$”)), ((polarizer, “$OptOut_1$”),(PMfiber, “$OptIn_2$”)), ((PMfiber, “$OptOut_1$”), (isolator, “$OptIn_2$”)), ((isolator, “$OptOut_1$”),(PMfiber, “$OptIn_2$”)), ((PMfiber, “$OptOut_1$”), (laser, “$OptIn_2$”))}

D. MS4ME models

The following optical components, shown in Table 8, and coupled models, shown in Table 9, were built and tested with the MS4ME simulation software during conceptual modeling research for the *qkdX* simulation framework.

Table 8. Coupled submodules.

Classical pulse generator	Polarization modulator	Decoy state generator	Classical to quantum	Optical security layer	Timing pulse generator	Optical power monitor

Table 9. Optical components.

Bandpass filter	Beamsplitter	Circulator	Optical photodiode	EVOA	Fixed attenuator	Half-wave plate	In-line polarizer	Isolator
Laser	Optical switch	PM fiber	Polarization controller	Pulse modulator	Polarizing beamsplitter	SM fiber	WDM	

EVOA: electronically variable optical attenuator; WDM: wave division multiplexer; PM: polarization-maintaining.