Characterising Enterprise Application Integration Solutions as Discrete-Event Systems

Sandro Sawicki
Unijui University, Brazil

Rafael Z. Frantz
Unijui University, Brazil

Vitor Basto-Fernandes
Polytechnic Institute of Leiria, Portugal

Fabricia Roos-Frantz
Unijui University, Brazil

Iryna Yevseyeva
Newcastle University, United Kingdom

Rafael Corchuelo
University of Seville, Spain

ABSTRACT

It is not difficult to find an enterprise which has a software ecosystem composed of applications that were built using different technologies, data models, operating systems, and most often were not designed to exchange data and share functionalities. Enterprise Application Integration provides methodologies and tools to design and implement integration solutions. The state-of-the-art integration technologies provide a domain-specific language that enables the design of conceptual models for integration solutions. The analysis of integration solutions to predict their behaviour and find possible performance bottlenecks is an important activity that contributes to increase the quality of the delivered solutions, however, software engineers follow a costly, risky, and time-consuming approach. Integration solutions shall be understood as a discrete-event system. This chapter introduces a new approach based on simulation to take advantage of well-established techniques and tools for discrete-event simulation, cutting down cost, risk, and time to deliver better integration solutions.

Keywords: Simulation, Discrete-Event Simulation, Enterprise Application Integration, Domain-Specified Language, Conceptual Model, Formal Model, Markov Decision Process, Queueing Theory.

INTRODUCTION

Frequently the current business processes in an enterprise has to be improved or new ones have to be created to support the enterprise's daily activities. All over the years, enterprises have been purchasing off-the-shelf software as well as developing custom applications aiming at giving computational support to implement and run their business processes. As a result it is not difficult to find an enterprise which has a software ecosystem composed of applications that were built using different technologies, data models, operating systems, and most often were not designed to exchange data and share functionalities. It is common that, from an administrative point of view, such software ecosystems are viewed as homogeneous systems, and are expected to support the enterprises' business processes by using current data and functionality without requiring too much investment. Enterprise Application Integration (EAI) provides methodologies and tools to design and implement integration solutions. The goal of an integration solution is to keep a number of applications' data in synchrony or to develop new functionality on top of them, so that applications do not have to be changed and are not disturbed by the integration solution (Hohpe & Woolf, 2003).

Amongst the state-of-the-art integration technologies available for enterprises to design and implement integration solutions, there are Camel (Ibsen & Anstey, 2010), Spring Integration (Fisher, Partner, Bogoevici, & Fuld, 2010), Mule (Dossot & D'Emic, 2009), and Guaraná (Frantz & Corchuelo, 2012). These technologies provide a domain-specific language that enables the design of conceptual models for integration solutions. Such languages follow the Pipes and Filters architectural style and provide support for using integration patterns (Hohpe & Woolf, 2003). In the Pipes and Filters style, a larger process is divided into a number of smaller and independent services (Filters), which are usually desynchronised by channels (Pipes). The integration patterns document a set of well-known services that are directly related to integration tasks.

The analysis of enterprise application integration solutions to predict their behaviour and find possible performance bottleneck is an important activity that contributes to increasing the quality of the delivered solutions. It is not enough to design a conceptual model for an integration solution, but it is also essential to analyse its behaviour under different workloads and minimise performance bottlenecks. Most often, the approach adopted by software engineers requires the construction of the integration solution, the execution of the actual integration solution, and the collection of data from this execution. This is a costly, risky, and time-consuming approach. The construction of integration solutions is expensive because it demands a good command on an integration technology to map the conceptual model into executable code. The execution involves the setup of a controlled environment in which the integration solution can be deployed, the generation and injection of input data into the integration solution, and the emulation of critical running scenarios. Any faults in the constructed solution may cause the execution to fail and negatively affect the analysis. The collection of data requires the insertion of extra code into the constructed integration solution and the configuration of the runtime system of the adopted integration technology that enables monitoring and collecting. A new approach that enables the analysis of the behaviour and discovering possible performance bottlenecks still in the design phase, taking as input the conceptual models of application integration solutions, would help to reduce cost, risk, and time spent.

Simulation is a research field that deals with experimentation of models to make predictions about the behaviour and the performance of actual systems. This chapter focuses on systems whose

models are classified as stochastic, dynamic, and discrete. Stochastic models use probability to model actual systems in which uncertainty is present, which makes them a good option to represent the real-world. Integration solutions can be characterised as stochastic systems because the arrival rate of their input data cannot be predicted and is essentially random, totally dependent on the applications that are being integrated. Dynamic models represent systems that change their state over time. The state in an integration solution changes as a result of the number of messages being processed in a particular execution time, and the operations executed on messages. Discrete models are event-oriented and so used to model systems that change their state at discrete moments in time from the occurrence of events. Integration solutions can be characterised as discrete systems for the reason that all components involved in an integration solution consume a particular execution time when an event occurs. Thus, any event can change the state of the integration solution. As a discrete system, the conceptual model designed for an integration solution can be simulated taking the advantage of well-established techniques and tools for discrete-event simulation, cutting down cost, risk, and time to deliver better integration solutions.

High-level state-based modelling languages available in these tools provide description syntaxes for model construction, ability to compute the reachable state space, indication if specific properties are satisfied by the model and other quantitative results relevant to identify interesting patterns or trends in the behaviour of a system. Model checking, simulation and experiments configuration support allows for the analysis of system properties as functions of model and property parameters.

The research work presented in this chapter aims at studying how to build a formal simulation model from a conceptual model designed using an application integration technology that supports the integration patterns proposed by Hohpe and Woolf (2003). Therefore, we characterise enterprise application integration solutions as discrete-event systems and show how their main building blocks can be modelled using examples of queueing theory and markov processes.

The recent work presented in Aalst (2015), describes the phases of a traditional simulation study. The author argues that the simulation study is composed of eleven phases, namely: problem definition, modelling, conceptual model, realising, executable model, verifying and validation, validated model, experimenting, simulation results, interpreting, and answers solutions. The problem definition concerns with the goals and what will and will not be part of the simulation. The next phase is modelling. Here, the conceptual model is created, as well as classes and the most relevant properties of objects. Besides, relationships between objects are defined. With the conceptual modelling completed, starts the realisation phase. In this phase, the conceptual model is mapped onto an executable model. The executable model created can be simulated on the computer, for this, depends on the simulation tool or simulation language used for its design. In addition to the verification, the validation of the model is necessary. The validation compares the simulation model with the actual system. With the validated model, experiments can be performed. In this phase the experiments have to be conducted efficiently in order to ensure reliable results. After, the simulation results have to be interpreted to allow feedback to the definition of the problem. Finally, a final report with answers to issues from the problem definition and proposals for solutions is created. These phases are illustrated in Figure 1. The scope of this chapter relates to the phases of the conceptual model and the executable model. The conceptual model used as example in this chapter was represented using Guaraná technology and the examples of simulation model were specified in PRISM modelling language for model checking and analysis (Oxford, 2014). This chapter starts describing the related work which has studied discrete event simulation aimed at discovering performance

bottleneck and some other works that have studied simulation in the context of business process. Next section, provides a background on simulation that helps readers to understand the fundamental concepts involved in this proposal. The following section provides a discussion on the characterisation of enterprise application integration solutions as discrete-event systems by studying the Guaraná technology and identifying which kind of information can be observed in a simulation of integration solutions, focusing on the prediction of their behaviour and finding possible performance bottlenecks. In the next section, an actual technique is presented and a software tool to support simulation is introduced. Last section, points out future research directions and concludes this chapter. The audience of this chapter are undergraduate and postgraduate students that are starting their researches in the simulation field focused on integration solutions and business processes.

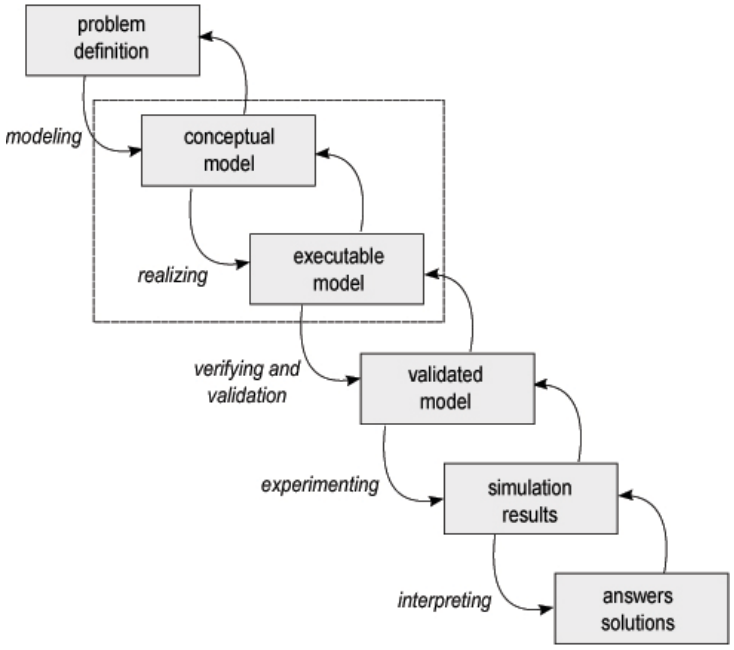

Figure 1: Phases of a traditional simulation study (Aalst, 2015)

RELATED WORK

There are some previous work in the literature that have used discrete-event simulation techniques and tools to analyse systems to predict their behaviour and find possible performance bottlenecks. Nevertheless, from the literature there is no evidence that discrete-event simulation has been explored aiming at the analysis of conceptual models of enterprise application integration solutions. There is a work that has used discrete-event simulation in the field of EAI, however in this work Janssen and Cresswell (2005) focus on organisational issues and the interests of stakeholders, by providing a simulation-based methodology to evaluate the impact of enterprise integration at business level before its implementation. They argue that the commitment of stakeholders is one of the keys for the success of EAI, and thus make stakeholders central to their methodology. Their research analyses integration problems only in the context of public organisations, in which they use discrete-event simulation to quantify the benefits that the integration could bring to the organisation at business level. The proposal

presented in this chapter differs from theirs since its focus is on the infrastructure technology used to design and implement integration solutions. Another work, presented by Al-Aomar (2010), describes the basic structure of service system simulation using application case studies targeting the performance of the service system. The author identifies and defines the main characteristics and elements of service system as system entities, service providers and customer service. In addition, he explores the modelling techniques that are used in the development of discrete-event simulation models targeting service systems. This work considers model elements, model logic, model data, model parameters, decision variables, and performance measures in its case studies simulation. The number of entities that arrive in the specified time interval is a random variable that follows Poisson distribution. Arrival rates and service rates are essential to calculate system performance measures. In this study the author collects the arrival and service rates, which are attributes used to represent customers arriving at a service centre. For example, for each customer, the times of arrival, service start, and service end are calculated. Having these times collected, the Time Between Arrival and Service Time can also be computed. Time Between Arrival is the time since last arrival and Service Time is the total time used by a service. After, Mean Time Between Arrivals and Mean Service Time are used to compute the average time. By using this strategy to collect inter-arrival and service times, it is possible to find bottlenecks located at services and queues. The article presented by Desa et al. (2013), demonstrates the potentiality of discrete-event simulation for detecting bottlenecks. The authors developed a discrete-event simulation model based on the logic and using data collected from a manufacturing plant specialised in producing aircraft parts. In this work, the detection of bottlenecks was based on resource utilisation and work in process. Arena simulation software was employed to model and analyse the several activities. With the model proposed in this study, it was possible to understand and improve the performance of the production system; furthermore, it was found that the discrete-event simulation is capable of analysing the behaviour of complex and sophisticated systems. Kunz et al. (2011) present a performance prediction methodology that calculates the best possible performance bound for expanded parallel discrete-event simulations. According to the authors, predicting and analysing runtime performance features understanding its behaviour is an important step in the development process of parallel discrete-event simulations. Their methodology is based on linear program that calculates an optimal event execution schedule for a given simulation and a set of microprocessors. They use linear programming for predicting the runtime performance of a parallel simulation model. A trace is used to the linear program that computes an optimal event schedule targeting to minimise the overall simulation runtime. Discrete-event simulation model can also be used to an automated bottleneck analysis to detect running production constraints. Faget et al. (2005) present a case study that was conducted in Toyota Company. The authors describe the application of a method for detecting bottlenecks using discrete-event models and design of experiments to suggest improvement alternatives. They also report that the main challenge was to educate the decision makers about the importance of simulation as a support tool to analyse the behaviour of production systems. The results obtained with the simulation had better accuracy in bottleneck analysis and provide useful information for improvements.

Simulation has also been studied in the context of business processes (Aalst, 2010; Aalst et al., 2010; Aalst, 2015; Hlupic et al.,1998; Hlupic, 2003; Jansen-Vullers et al., 2006 and Rosinat et al., 2009). Recently, in Aalst (2015), the author discusses potential pitfalls involved in the traditional simulation approaches, which can lead to bad results. In this research, he introduces traditional business process simulation using the description of the simulation-specific elements of process models and discusses the

different phases of a typical simulation project. Furthermore, he proposes the use of simulation as an analysis tool for business process management and the construction of simulation (formal) models using mining techniques, which take as input real event data by focusing on validation of simulation models, correct derivation and interpretation of simulation results. In Aalst (2010), the author argues that the use of simulation is often limited in real world and typically fails in companies. According to the author, the simulation is considered a relevant and highly applicable tool in Business Process Management, however, few organisations effectively use simulation. He explains that in many situations the quality of the simulation model leaves much to be desired, because, essentially, there are three problems. The first one occurs when the process is modelled incorrectly. The second happens when there is not enough was data collected to be able to parameterise the model, and the third problem occurs when the language does not allow for the modelling of more detailed behaviours. He then discusses the limitations and point out his ideas to overcome the problem. In Aalst et al. (2010), the authors go deep and study the problem of modelling in a naive manner the behaviour of resources involved in a business process. They explain that the practical relevance of business process simulation is limited, because often the simulation results do not match with reality. Moreover, the authors show through experiments that is possible to get the human behaviour in business processes effectively. The approach summarises the main pitfalls in this process and addresses one particular problem related to the availability of resources. In Hlupic and Robinson (1998), the authors explain that the competitiveness caused by globalisation incites many organisations to change their business processes. This procedure can provide significant benefits such as costs reduction or improving efficiency, however, there are risks associated with it. They report that modelling and analysis of business processes using simulation can reduce that risk and present an investigation of potential of simulation modelling that can be used for modelling business processes. Furthermore, this approach presents a discussion of simulation models that represent business process. In Hlupic (2003), the author reports that many strategies of change management has been explored for organisations, this is due to the need to adapt to new business conditions. This paper explain that many studies suggest that the success of projects could be improved using development of dynamic models of business processes without the need for a radical change. Besides, the author argues the importance of wider use of simulation techniques by business community and investigate the usability of simulation modelling for evaluating the alternative business process strategies. In Jansen-Vullers and Netjes (2006), the authors discuss several simulation tools that are relevant for the Business Process Management area and explain the criteria used for its evaluation. Furthermore, they evaluate their applicability for Business Process Simulation and make recommendations for future research. In Rosinat (2009), the authors now focus on the use of process mining techniques based on process historic data that would help, for instance, to evaluate the performance of different alternative designs. However, in this chapter, the approach does not have real event data, since the simulation models have to be created during the design phase of the integration solution, in which there is no constructed solution yet. This scenario aim at building a simulation model from an application integration conceptual model, so that the conceptual model can be analysed and possibly improved to solve performance problems.

BACKGROUND

In this research, simulation refers to simulation of a system. According to Forrest (1968), a system is a grouping of parts that work together, seeking a common goal. It presupposes an interaction of cause-effect between its parts. A system can be studied using experiments with actual models, or

experiments with models of actual systems (Babulak & Wang, 2010). In practice, what is meant by the system depends on the target of a specific study (Law & Kelton, 1991). Figure 2 illustrates the different ways to study a system.

Simulation is a quite generic term, since there are several strategies and mechanisms for representing the systems behaviour. Simulation can be considered as the study of the behaviour of actual systems by experimentation of models. Generally, simulation is recommended when problems are impossible or expensive to be solved by actual experimentation or when problems are highly complex to be treated analytically. In view of the fact that simulation considers the stochastic characteristics of a system, it can reproduce system behaviour with great realism and fidelity.

Simulation can be classified in two categories, namely: non-computational and computational (Chwif, Paul & Barreto, 2006). The non-computational simulation uses a physical model to obtain information about the system behaviour, such as the prototype of an automobile, which can be included into an air chamber to obtain information about its aerodynamics. The computational simulation is a process that describes a computational model from an actual system. It can conduct experiments with this model targeting the understanding of its behaviour and/or evaluate strategies for its operation. The simulation does not predict the future, but can predict the behaviour of a system based on its input data and a set of rules. According to Aalst (2015), the use of simulation can present some disadvantages, namely: (i) the simulation study can be time consuming, moreover, in some situations, the reliability of the results require long simulation time; (ii) the results obtained by the simulation has to be carefully interpreted. The results have to represent reality; (iii) the simulation does not provide proofs. Situations that could appear in reality do not occur during the simulation experiment.

Simulation models are required to simulate a system. For example, given a system, it is possible to construct a simplified model to represent the interactions between its parts. In this context, a model is an abstraction of the reality, approaching the true behaviour of the system, but always simpler than the actual. It can be difficult to achieve this characteristic because realistic models are rarely simple, and simple models are rarely realistic.

Simulation models better capture the actual behaviour of systems and allow them to be analysed on computers. They are used to get answers about the operation of the system and may be able to analyse several aspects of it, or rather, all that are of interest. Actual systems generally have a higher complexity, mainly due to their dynamic and random nature.

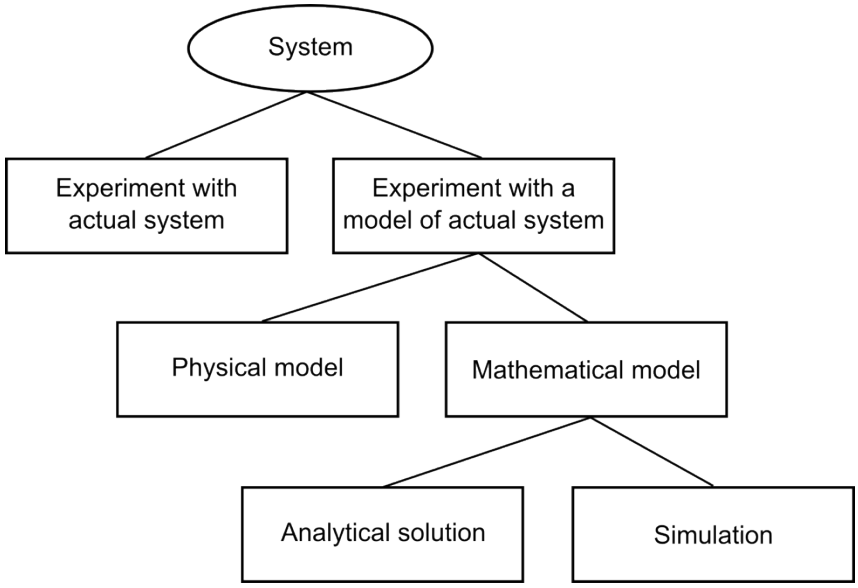

Figure 2: Ways to study a system (Law & Kelton, 1991)

According to Paul and Balmer (1993), the development of a simulation model is done in three stages: model conception, model implementation, and analysis of the results. In the first stage, software engineers must clearly understand the system to be simulated, its goals, its assumptions, its input data, and discuss the scope of the model. In this stage, a well-known technique shall be used to represent the simulation models, so that other people involved in the project can understand it. The second stage transforms the conceptual model into a computational model, by means of a simulation language or a simulator. The third phase performs the simulation, analyses its results and documents it. If necessary, the model can be modified and the cycle restarted.

Models can also be classified as symbolic or analytic. Symbolic models are composed of graphic symbols that represent a system in a static way, for instance, a picture or a flowchart. This kind of model has limitations, because there is a lack of quantitative information and difficulties to represent details (Chwif, Paul & Barreto, 2006). According to Harrel and Tumay (1994), analytic models are often exemplified as a set of mathematical formulas. Most often, these models are static and do not present analytical solutions for complex systems. In both cases, simplifying assumptions should be used.

Figure 3 illustrates the model taxonomy proposed by Law and Kelton, (1991). This classification uses either deterministic or stochastic models. If a simulation model does not contain any probabilistic variables, it is called deterministic. In a deterministic model, for a known set of data input the result will be a unique set of output. A stochastic or probabilistic model uses one or more random variables as input that lead to random outputs. The output of stochastic simulation should be treated as statistical estimation of the actual characteristics of a system. Typically stochastic models are more complex and better represent actual systems than deterministic models.

Static models represent a system at a particular time. They do not change over time and therefore can be used in other fields, such as in numerical methods, optimisation problems, test of algorithms, and Monte Carlo models. They are also applied in probabilistic simulation algorithms, such as Simulated

Annealing (Kirckpatrick, Gelatt & Vecchi, 1983). On the other hand, dynamic models represent systems that change over time. The majority of the simulations use dynamic models, because most of the systems change over time.

According to Law and Kelton (1991), discrete or continuous simulation models can be defined analogously to the way of discrete and continuous system. Then, the continuous simulation models are used to model systems in which the state variables change continuously with respect to time (i.e., the system state changes continuously in accordance with time). The discrete-event simulation is event-oriented. It is used to model systems that change their state at discrete moments in time from the occurrence of events. In some cases it may be necessary to build a simulation model that understand, simultaneously, aspects of continuous and discrete simulations. It is known as "hybrid or combined simulation" (Pritsker, 1986). The decision whether to use a discrete or a continuous model for a particular problem depends on the specific objectives of the study.

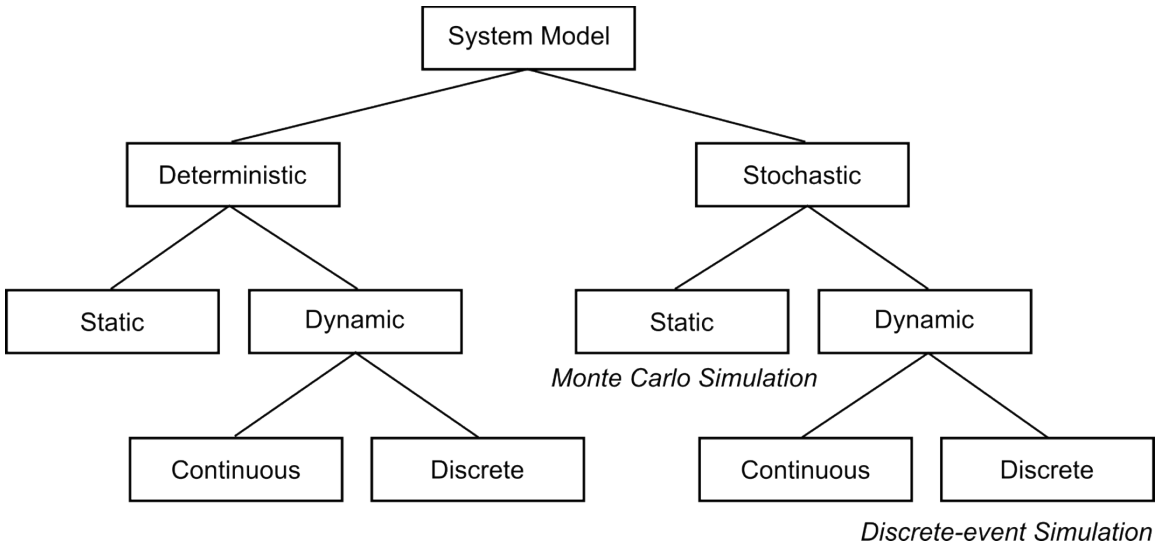

Figure 3: Model Taxonomy (Law & Kelton, 1991)

There are three basic elements that compose a discrete-event simulation model, namely: entity, attribute, and activity. Any object involved with the model is called entity. The property of this entity is named attribute. Any process that generates a change in the model is called activity. For example, in a discrete-event simulation model of a bank, entities would represent customers that arrive into the system; attributes would be their balance and available personal credit, and the service provided by a specific department of this bank, an activity.

A model for discrete-event simulation aims to reproduce the activities of entities involved in the scope of the system in order to know something about its behaviour and performance. Thus, it is necessary to define the states of a system and events that change the system from one state to another. The state of a system can be defined as a set of variables used to describe the behaviour of the system at a particular time. This set is called state variables. In the banking system, the possible state variables would be the number of customers and the time of arrival of each customer in the bank (Law & Kelton, 1991). In a discrete-event simulation, each event occurs at a particular moment in time and determines a change of

state in the system (Robinson, 2014). A process is an ordered sequence of events and can include several activities.

There are three ways that can be used to formulate models for discrete-event simulation: (i) taking as input the specification and description of the process used by the system entities; (ii) taking as input the complete description of activities of the entities involved in the system, and (iii) by defining changes that can occur in the states at each time of event.

The simulation model would be implemented using strategies of simulation. According to Pidd (1998), there are four basic strategies to simulate discrete-event systems, which are based on event, activity, process, and a combination of the first two strategies, named three-phase method. In the event strategy, it is possible to specify the particular time actions occur in the model. In the activity strategy, it is possible to specify reasons, based on preconditions, for the actions to occur in the model. In the process strategy, it is possible to specify the sequence of actions that build the flow of each entity in the system. In the three-phase method, phases are called A, B, and C events. The second phase (B-events) searches for completion of activities. The third phase (C-events) finds the conditions so that these activities can occur. Then, the first phase (A-events) updates the time, advancing the system to the next event.

INTEGRATION SOLUTIONS AS DISCRETE-EVENT SYSTEMS

Characterising a conceptual integration solution as a discrete-event model enables to predict the future behaviour of the system by preventing the effects produced by changes in this system. Thus, it becomes possible to construct theories and hypotheses based on observations conducted through the models. Furthermore, time can be controlled, in other words, it can be compressed or expanded, allowing to reproduce phenomena of slow and fast way in order to understand and study the behaviour of the system.

A discrete-event model can identify and reproduce the most important variables related to performance and how they interact with each other and with other elements. This makes it easier to understand the behaviour of the system. This section starts by introducing Guaraná technology and then characterises enterprise application integration solutions as a discrete-event system. It also provides some discussion on how simulation can be performed on this kind of system.

Guaraná

Guaraná technology supports software engineers in the design, implementation, and execution of integration solutions. Since the focus of this chapter is on the simulation of conceptual models, this section aims to provide a brief overview of the modelling language provided by this technology. In Guaraná, the conceptual models are designed using a domain-specific language, which has an easy-to-learn and intuitive graphical notation. This language is based on the integration patterns documented by Hophe and Woolf (2003) and enables the design of platform-independent models, i.e., the resulting models are not committed to a particular implementation technology (Frantz et al., 2010). The following concepts are supported by constructors in Guaraná language:

- **Message:** An abstraction of a piece of information that is exchanged and transformed across an integration solution. It is composed of a header, a body, and one or more attachments. The header includes custom properties and frequently the following pre-defined properties: message

identifier, correlation identifier, sequence size, sequence number, return address, expiration date, and message priority. The body holds the payload data, whose type is defined by the template parameter in the message class. Attachments allow messages to carry extra pieces of data associated with the payload, e.g., an image or an e-mail message.

- **Task:** Represents an atomic operation that can be executed on messages, such as split, aggregate, translate, chop, filter, correlate, merge, resequence, replicate, dispatch, enrich, slim, promote, demote, and delay. Roughly speaking, a task may have one or more inputs from which it receives messages, and one or more outputs by means of which messages depart. Depending on the kind of operation, a task may be stateless or stateful. In a stateless task, the completion of its operation does not depends on previous or future messages; contrarily, the operation of a stateful task depends on previous or future messages to be completed, such as the case of the aggregator task, which has to collect the different correlated inbound messages to produce a single outbound message. In this chapter, stateful tasks are not considered because the vast majority of tasks used in integration solutions are stateless.

- **Slot:** A buffer connecting an input of one task to the output of another task aiming at messages to be processed asynchronously by tasks. A slot can follow different policies to serve messages to tasks, such as a priority-based output or a first-come, first-served. If a priority is defined in the message, slots follow the former policy; otherwise, the latter policy is adopted. In this chapter, it is assumed that messages have no priority and the slot serves them in a first-come, first-served policy.

- **Port:** Abstracts away from the details required to interact with an application within the software ecosystem. Roughly speaking, by means of a port it is possible to establish read, write, solicit, and respond communication operation with the applications being integrated.

- **Integration Process:** Contains integration logic that executes transformation, routing, modification, and time-related operations over messages. An integration process is composed of ports that allow it to communicate with the applications being integrated, slots and a set of tasks to specify the integration logic.

Conceptually, an integration solution aggregates one or more integration processes through which messages flow and are processed asynchronously. The integration flow is actually implemented as a Pipe and Filter architecture, in which the pipes are implemented by Slots and the filters are implemented by Tasks. Every task realises an integration pattern (Hophe & Woolf, 2003) and its execution depends on the availability of messages in all slots connected to its inputs. Slots are key constructors to enable asynchrony in an integration solution, thus messages are stored on them until they can be read by the next task in the integration flow. A detailed discussion on the domain-specific language provided by Guaraná is presented in Frantz et al. (2011).

Figure 4 introduces a conceptual model designed using Guaraná to solve the Café integration problem introduced by Hophe (2005). The Café case study has become the de facto standard to introduce an integration technology from a practical point of view. The workflow in Café describes how customer orders are processed in a coffee shop. Roughly speaking, it starts by a customer placing an order to the cashier, who then registers the order in the system and adds it to an order queue. An order may include entries for hot and cold drinks, which are prepared by different baristas. When all of the drinks that

correspond to the same order have been prepared, they are ready to be delivered by the waiter. Every order has a tray associated to it, which is used to deliver the order to the customer. Note that, the cashier is decoupled from the baristas, since the orders taken from customers are placed in the queue from which baristas retrieve them. It allows the cashier to keep taking orders from customers even when the baristas are backed up. Baristas do not have a complete view of the whole set of drinks in an order, they receive individual drink requests and when a drink is prepared the barista places it on the corresponding tray. The goal of the integration solution is to take orders from the order queue, send requests to the corresponding baristas to prepare the corresponding hot and cold drinks, and notify the waiter when an order is completed.

The integration solution is composed of one integration process that exogenously co-ordinates the four applications involved. The communication with the Orders and Waiter applications is carried out by means of ports that read and write messages; on the other hand, the communication with Barista Cold Drinks and Barista Hot Drinks is carried out by means of ports that interact with API that these applications export. The workflow begins at entry port P1, which periodically reads the Orders application log to find new customer orders. Every order results in a message with the drinks to be prepared added to slot S1. Task T1 splits every message into several other messages, one for each drink. Messages are now routed by the dispatcher task T2 either towards the Barista Cold Drinks or Barista Hot Drinks applications. Task T3 replicates messages to the Barista Cold Drinks, so that one copy can be used to request this application to prepare the drink, by means of port P2, and the other waits for the correlated response that brings information about the preparation of the drink. Task T5 correlates the response from the barista with the waiting copy, and task T6 enriches the waiting copy with the information returned by Barista Cold Drinks. Task T4 transforms messages into the necessary request format for the application. Messages towards Barista Hot Drinks behave symmetrically. Once the drink is prepared, the messages from baristas are merged into a single slot S2 by the merger task T7. Then, the prepared drinks are taken from this slot and aggregated back into a single message for the order, so that exit port P4 writes the resulting message to the Waiter application.

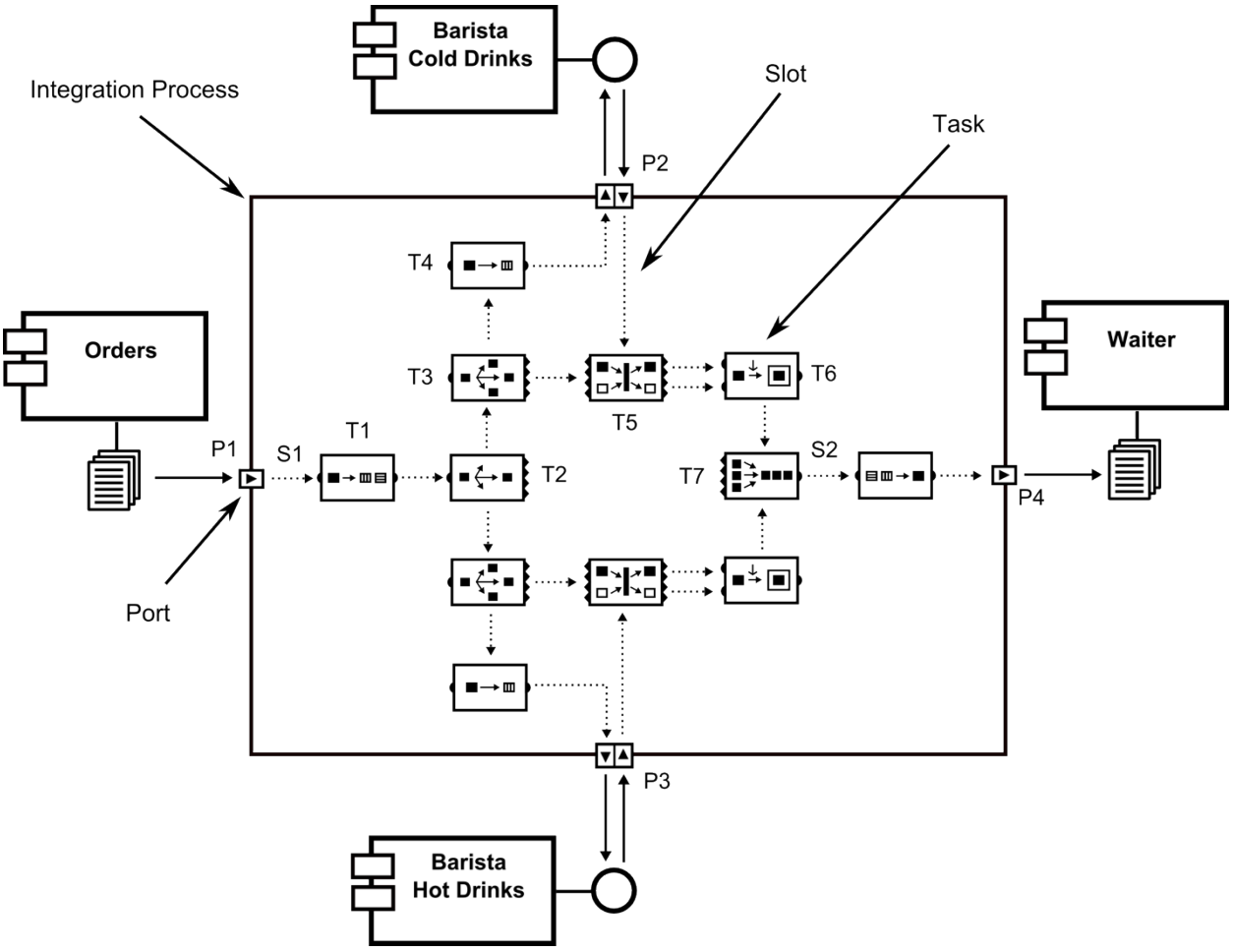

Figure 4: A conceptual model for the Café integration solution.

Characterisation

In the modelling of discrete-event systems, such as an integration solution, several paradigms can be considered. Some of these paradigms are: queueing theory (Gross et al., 2008; Kleinrock, 1975), Markov chains (Cao & Ho, 1990; Bremaud, 1998), semi-Markov process (Janssen & Manca, 2006), Petri nets (Petri, 1966), formal languages and automata (Hopcroft & Ullmanm 1979; Klein, et al., 2014), and some specific formalisations as max-plus algebra (Bacelli et al., 1992) and min-plus algebra (Cassandras & Lafortune, 1999).

Figure 5 (a) illustrates a generic model of a service system. This process is based on queuing models, which have a similar structure to the conceptual model of an integration solution designed with Guaraná. In this structure, customers arrive to the queue, and wait a particular time to be served. Similarly, Figure 5 (b) illustrates an excerpt of an integration solution designed with Guaraná, in which the messages represent customers, slots represent queues, and tasks represent services.

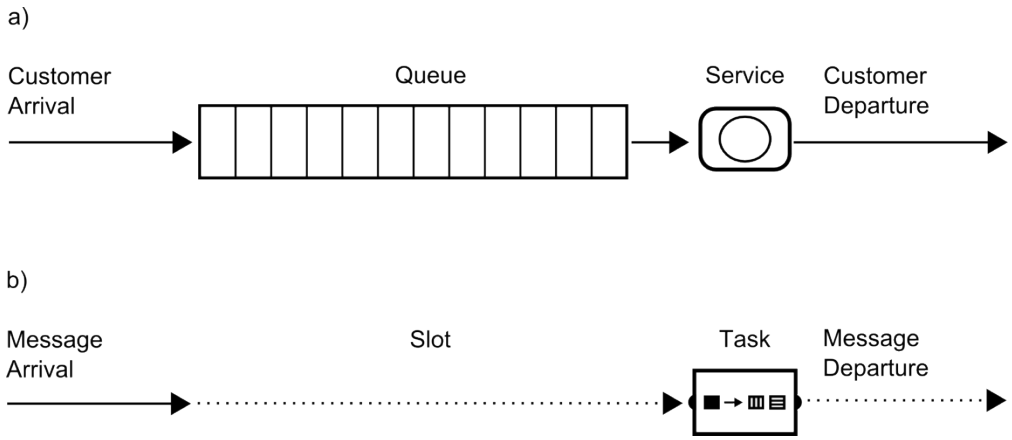

Figure 5: A generic model of a single service system (a) using Guaraná (b)

In an integration solution, the arrival rate of messages is an important random variable. The symbol λ is used to quantify this variable and means the average rate of arrivals. The variable *Ar* represents the mean interval between arrivals. For example, if 60 messages arrive per minute (λ), then the mean interval between arrivals (*Ar*) is one second.

The service process of an integration solution is also quantified by a random variable. The symbol μ indicates the mean rate of service, and the variable *Se* is used for the average execution time of a task (service). For example, if a task processes 120 messages per minute (μ), then each task would spent 0.5 seconds per message.

Another common characteristic between the integration solution presented by Guaraná and a discrete-event system is the relationship between its elements and operation structure. Figure 6 illustrates the relationship between conceptual elements of a discrete-event system. It is possible to relate them to elements of an integration solution designed with Guaraná. For example, Figure 6 shows the entities arriving to the system. In Guaraná the entities are messages, i.e., objects that allow the interaction with the system. A process is a sequence of activities. An activity occurs between two events and, during an event, the state of the system can change. The occurrence of an event in Guaraná can be characterised as the arrival of a message to be processed by a task. Every time a task processes a message, the system changes its state. A task occurs between two events, so it can be characterised as an activity, and consequently a process can be characterised as a sequence of tasks.

The discipline of slots is another important feature in an integration solution because it defines how the next message will be processed by the task. Several policies can be used at slots, such as first-in-first-out, last-in-first-out, service in random order, priority policy, shortest job first, and shortest remaining processing time. In general it is also possible to define the size of queues, however, in an integration solution designed with Guaraná, the queue size (characterised in this chapter as slot) can contain an infinite number of messages.

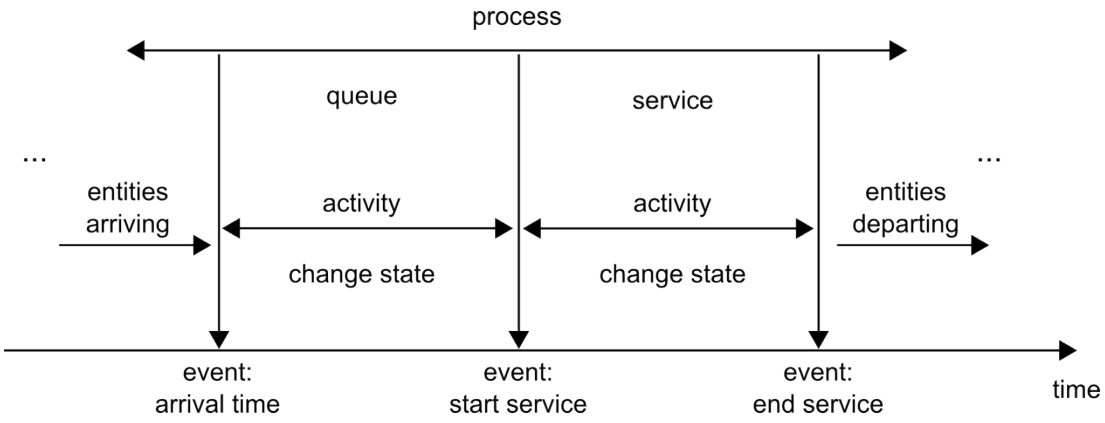

Figure 6: Relation between entity, event, state, process and activity

Inter-arrival and task times

In an integration solution, arrival rates and task rates are fundamental to calculate system performance measures, such as average waiting time, utilisation, and average time in system. Some single attributes that can be used to collect the arrival and task rates are: Arrival Time (AT), Time Between Arrival (TBA), Task Start Time (TST), Task End Time (TET), and Task Time (TT).

For each message, it is possible to collect the times of arrival, task start, and departure time (task end time). Based on the work presented by Al-Aomar (2013), it is possible to calculate Time Between Arrivals and Task Time. These variables are used to find Mean Time Between Arrivals (MTBA) and Mean Task Time (MTT).

For example, if the sum of the Time Between Arrivals is 100 seconds to process 1,000 messages, it means that each message waited 0.1 seconds (on average) in the slot for a task (MTBA = 0.1 seconds). If the sum of the Task Time is 80 seconds to process 1,000 messages, it means that it is necessary 0.08 seconds to process each message (MTT = 0.08 seconds). Thus, the service rate can be defined as $\mu = 1/MTT$, and mean arrival rate as $\lambda = 1/MTBA$.

The arrival process

The arrival process indicates the arrival standard of messages in an integration solution. It presents stochastic behaviour, i.e., arrivals occur probabilistically in time and space. Thus, it is necessary to know which probability distribution describes the time between arrivals of messages. There are several probabilistics distributions used to discrete-event simulation, such as Poisson distribution (Haight, 1967), Boltzmann distribution (DiPerna & Lions, 1989), negative binomial distribution (Hilbe, 2011), hypergeometric distribution (Tohma et al., 1991), and negative exponential distribution (Schmidt & Makalic, 2009).

Poisson distribution is a discrete probability distribution that can be applied to describe the number of occurrences of events in a given time interval. The basic assumptions for the use of Poisson distribution are: (a) the conditions of experiments remains constant over time, i.e. the mean occurrence rate is constant over time; (b) disjoint time intervals are independents, i.e. the information about the number of occurrences in an interval of time does not affect the number of occurrences in another interval

of time. The poisson distribution has proven applicable to several types of arrivals processes related to real systems and thus its use is widespread in modeling queues.

According to concepts of statistical mechanics, physics and mathematics, the Boltzmann distribution (known as the Gibbs measure) is a distribution function, probability measure, or frequency distribution for the distribution of the states of a system based on canonical ensemble. The negative binomial distribution is a discrete probability distribution. It is a generalisation of the geometric distribution, in which the random variable is the number of Bernoulli samples required to obtain n successes. Bernoulli trial is a random experiment that generates two possible outcomes that are success and failure. Hypergeometric distribution is a discrete probability distribution that describes the number of success in n extractions of a finite population, without replacement. It refers to experiments that are characterised by withdrawn without replacement, in which the probability of success changes on each withdrawal. The geometric distribution differs from the binomial distribution, because the probability of success changes for each new experiment. Negative exponential distribution or exponential distribution is the probability distribution corresponding to Poisson distribution when considers intervals between arrivals, i.e. to analyse a process that follows the arrival rate of Poisson distribution, it is possible to verify that the intervals between arrivals will follow the negative exponential distribution. In continuous-time stochastic processes, the exponential distribution is widely used to model the time until an event occurs in the process. Furthermore, the exponential distribution is the most important continuous distribution for understanding and development of continuous-time Markov chains.

Time of tasks

The probability distribution of the time of tasks can be the same presented in arrival process. The service can be an independent state, because it does not depend on the number of messages waiting for a task. However, in a dependent state, the process of service can change in relation to the number of messages in the slot.

The Erlang distribution (Jodrá, 2012) can be used to model the time to complete a sequence of n events, such that each event requires an exponential period of time to complete. It is a probability distribution with wide applicability, mainly due to its relationship with the exponential distribution. For example, when a Poisson process is in operation, the Erlang distribution can be used to predict waiting times in queuing (slots). Originally, this method was developed to analyse the number of telephone calls that could be made simultaneously to the operators of the switching stations. Currently, this distribution is used in several areas that apply stochastic processes and it is a special case of the Gamma distribution (Choi & Wette, 1969).

Queuing models as slot models

Based on queueing theory, Kendall (1953) proposes a standard system that was used to describe and classify a queueing node. The model can be described as A/X/c/K/N/Z, such that A is the arrival process (distribution between arrivals), *X* is the service time distribution, c is the number of servers, *K* is maximum number of elements the system can handle, *N* is the population and *Z* is the discipline of the queue. This structure is similar to that used by slots and tasks.

Example of an integration solution based on the notation of a standard system proposed by Kendall (1953).

M/E/1/∞/∞/FIFO

Example 1:

- Arrival Poisson
- Erlang distribution (service time)
- 1 task
- Maximum number of elements = ∞
- Arrival messages (population) = ∞
- First-in-First-out (FIFO)

Example 1 describes the configuration of a process between messages, slot, and task based on Kendall's Theory, where *M* corresponds to the Poisson arrival process; *X* is Erlang distribution time; the variable *c* determines that the integration solution has only one task for each slot; *K* represents that the number of elements supported by the system is infinite; *N* determines that the number of messages waiting for entry into the system is infinite (population) and the discipline *Z* is FIFO.

The M/M/1 is another model based on queuing theory that is widely used for discrete-event simulation. It also allows the construction and understanding of the basic ideas and methods of queuing theory. In this model, arrivals are determined by Poisson process with mean rate λ (symbol *A* at Kendall's notation), and service times are exponentially distributed with rate μ (*X* symbol at Kendall's notation). There is only one server, the size of queue is infinite, and its discipline is FIFO. Furthermore, this model considers the cases of infinite and finite population. The state of the system is determined by the number of messages that are located in the system at a particular time (located at slot or tasks). Statistically, all messages are considered equal. An extension of the M/M/1 model that considers more than one server is the M/M/c model, such c is the variable that represents the number of servers.

In an integration solution, the M/M/1 model allows to obtain useful information about the behaviour of the system, such as: What is the mean number of messages in the slot? What is the mean number of messages in the system? What is the mean time that messages are waiting in the slot? These questions can be represented for some fundamental random variables, as shown in Table 1.

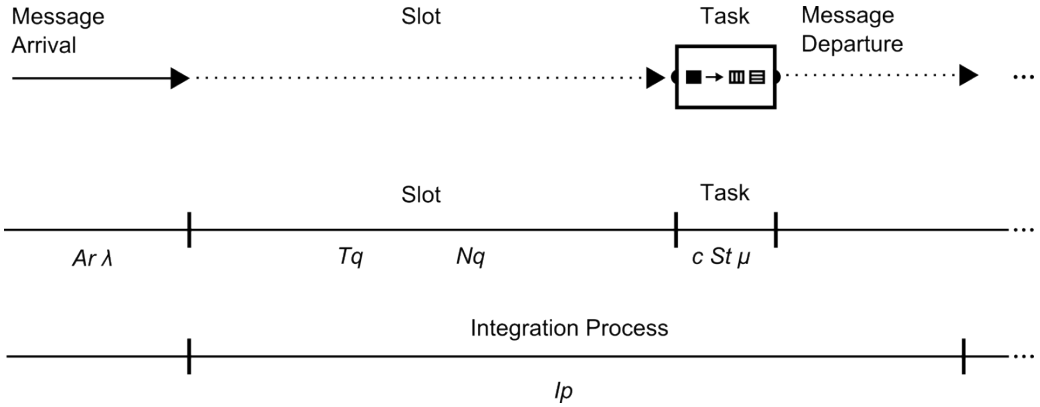

Figure 7: Location of variables in a single server model

Table 1: Random variables to evaluate the behaviour of simulation models

	Variable	Description	Equation
Concepts	λ	mean arrival rate	$\lambda = 1/MTBA$
	μ	mean rate of service	$\mu = 1/MTT$
	c	number of tasks that read from a single slot	* $c = 1$
Integration Solution	Sn	mean number of messages in the system	$Sn = \frac{\lambda}{\mu - \lambda}$
	Ip	mean time that the message stays in the system	$Ip = \frac{1}{\mu - \lambda}$
Arrival	Ar	mean interval between arrivals	$Ar = \frac{1}{\lambda}$
Slot	Tq	mean time spent by the message in the slot	$Tq = \frac{\lambda}{\mu(\mu - \lambda)}$
	Nq	mean of messages in the slot	$Nq = \frac{\lambda^2}{\mu(\mu - \lambda)}$
Task	St	mean time of service	$St = \frac{1}{\mu}$
	Nm	mean number of messages executed	$Nm = \frac{\lambda}{\mu}$

* The number of servers is a feature that depends on the execution model implemented by the integration technology. This chapter focuses conceptual models that are platform-independent, thus it assumes *c* is always one.

Figure 7 represents a model M/M/1 with a single slot for arrival messages (customers), and a single task (service). At a particular moment in time, the following events occur: events of arrival messages in slot (arrival time), events that start the service time (execution of task), and also events that indicate the end of service time (task concluded). The arrival process can be represented for variables λ and *Ar,* such that λ represents the mean arrival rate and *Ar* the mean interval between arrival. The variables *Tq* and *Nq* provide information about the behaviour of messages in slot. *Tq* provides the mean time spent by the message in the slot and *Nq* reports to the means of messages in the slot. The behaviour of tasks can be measured by variable *St* that represents the mean time of service, and also by variable μ that indicates the mean rate of service. The integration process is represented by variable *Ip* that shows the mean time that the message stays in the system.

The queuing theory is an analytical method that approaches problems of a system using mathematical equations. The dynamics of the operation of an integration solution is similar when compared to those adopted by the queuing theory. For example, a customer can be seen as a message, and it can wait in queue (slot) at a particular time to be processed by service (task). Since the characteristics of

the individual components of an integration solution are known, a simulation based on discrete-event is a good strategy to obtain quantitative and qualitative measures of the system. Thus, this section has explored the queuing theory in order to characterise elements of an integration solution as a discrete-event system.

MARKOV PROCESSES BASED SYSTEM MODELLING

In the previous section, an integration solution was addressed as a mathematical model of waiting queues, providing an in-depth view of the solution dynamics with respect to queue lengths and waiting times prediction. Queueing theory is commonly used in decisions support for resources usage prediction and planning. In this chapter, other formal modelling approaches are also addressed. Firstly we introduce modelling based on probabilistic extensions of temporal logics, aiming at the characterisation of integration solution as solutions that exhibit stochastic behaviour, operate under constraints on timing and other resources. Secondly, extensions of process algebras are analysed with emphasis on systems performance evaluation. Finally, Petri Nets based modelling is also considered and compared with other approaches. Corresponding modelling tools are presented for functional verification and performance analysis purposes.

A stochastic process represents in our context the evolution of a system of random variables over time. A process with some degree of uncertainty, that evolves through a set of possible states.

Stochastic model simulation in general and model checking in particular are important components in the design and analysis of software systems with probabilistic, nondeterministic and real time characteristics. Model checkers analyse if a system satisfies a state/transition model and a temporal logic specification. In addition, stochastic model checking calculates the likelihood and temporal relationships of certain events during system execution (Kwiatkowska et al., 2011).

Amongst several existing probabilistic models, discrete-time Markov chains, which allow to specify the probability of making a transition from one state to another, continuous-time Markov chains, which allow to model continuous real time and probabilistic choice, and Markov Decision Processes, which allow for both probabilistic and nondeterministic modelling, are of special interest for integration solutions formal modelling. Markov based simulation and model checking allows for systems properties analysis, such as path-based, transient and steady-state properties (Parker, 2011).

A Discrete-time Markov Chain can be described more formally as *a tuple of a finite set of states* (S), *an initial state* s_0 belonging to S, *a transition probability matrix* (P) of SxS→[0-1] with transitions sum from a state equal to 100% and a *labelling function* assigning *atomic propositions* ($L : S \rightarrow 2^{AP}$) to states. Time is discrete (discrete time-steps), homogeneous, and transition probabilities independent of time. Executions of Discrete-time Markov Chains are paths, with length equal to path transitions. Paths probabilities calculation allow for behaviour analysis, probabilistic reachability, quantitative and qualitative properties such as repeated reachability and persistence. *Reward structures* representing benefit or cost (Transition rewards - instantaneous and State rewards - cumultive) are modelled by reward functions $rt : S \times S \rightarrow R_{\geq 0}$ and $rs : S \rightarrow R_{\geq 0}$. A reward structure can also measure the time-steps spent in a

state or the chance of being in a state after a specific number of time-steps (Kwiatkowska et al., 2007; Parker, 2011).

Continuous-time Markov Chains main differences from Discrete-time Markov Chains models rely on transitions occurring in real time, being represented in a transition rate matrix ($R : S \times S \rightarrow R_{\geq 0}$) and assigning rates to pairs of states, instead of probabilities to transitions between states. In case there is more than one state s' with $R(s, s')>0$ (race condition), the first transition triggered determines the next state of the model. Paths are sequences of states with time attributes (states execution duration). *Transient* and *steady-state behaviour* properties address the likelihood of being in a state at a specific time (or in the long-run), and time spent in specific states. Reward structures are used to analyse number of requests served or queue sizes in a time interval, at any time instant or in the long-run (Kwiatkowska et al., 2007; Parker, 2011).

Markov decision processes follow the same time modelling of Discrete-time Markov chains and extend Discrete-time Markov chains fully probabilistic modelling by allowing for nondeterministic choice (Parker, 2011).

Formally, a Markov decision process is a tuple (S, s_{init}, Steps, L) where S is a finite set of states, $s_{init} \in S$ the initial state, $Steps : S \rightarrow 2^{Act \times Dist(S)}$ the transition probability function, Act a set of actions, Dist(S) the set of discrete probability distributions over S, and $L : S \rightarrow 2^{AP}$ a labelling with atomic propositions. A path is a sequence of states and action/distribution pairs, e.g., $s_0(a_0,\mu_0)s_1(a_1,\mu_1)s_2$, representing a system execution. Paths resolve nondeterministic (turn the model into a Discrete-time Markov Chain) and probabilistic choices, and then probability measure over paths are calculated (Kwiatkowska et al., 2007; Parker, 2011).

Figure 8 shows a Markov decision process model of a simple communication protocol, emphasising probabilistic and nondeterministic choice modelling differences. Process starts trying to send a message, followed by a nondeterministic choice between wait (if channel busy) and send (when channel is free). If send action occurs, probability of successful delivery is 0.99 and probability of failure is 0.01, followed by the corresponding stop and restart actions.

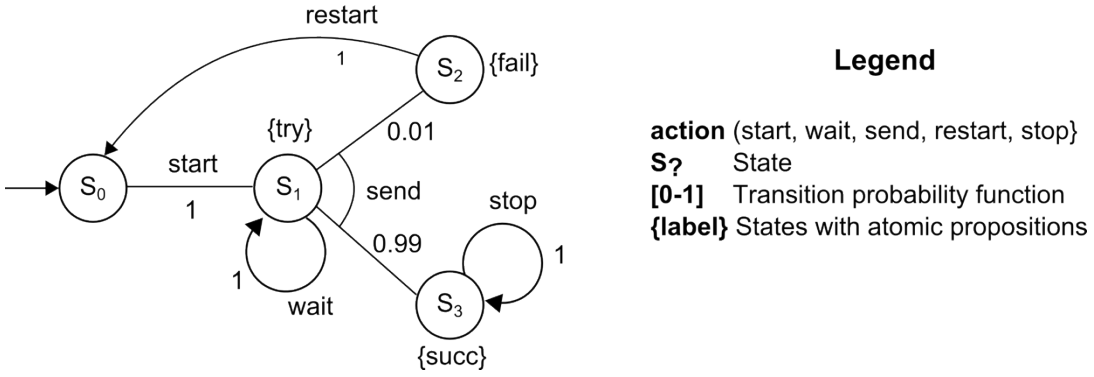

Figure 8: Markov decision process model of a simple communication protocol (Parker, 2011)

In Markov decision processes, adversaries (strategies or policies) are used to resolve nondeterministic choices (multiple distributions) and may belong to different classes such as memoryless, finite-memory, randomised, fair. An adversary is a function mapping every finite path to an element of the model Steps. Markov decision processes provide best/worst case analysis based on lower/upper bounds on probabilities over all possible adversaries (Kwiatkowska et al., 2007; Parker, 2011).

For integration solutions formal model checking and performance analysis, Markov processes might be of help. This type of probabilistic modelling has extensive support on modern probabilistic model checker software tools such as PRISM (Oxford, 2014).

A simplified and partial Markov decision process model of the integration solution presented in Figure 4 is shown in Figure 9, to illustrate schematically the modelling ability of Markov decision processes in the EAI domain.

This model can be specified in PRISM modelling language for model checking and analysis. Markov decision process concepts are directly supported by PRISM modelling concepts and are introduced next. The concept of *module* in PRISM represents system processes, including process *variables*, which describe system *states* and *commands* (process behaviour, i.e., the way states change over time) comprising a *guard* (condition referring to variables of this or other module, required for the update to take place) and one or more *updates* together with the corresponding probabilities (updates can only affect variables belonging to the module). A PRISM model is constructed as the parallel composition of its modules. Variables can be of *Boolean*, *Integer* or *Clock* type. PRISM provides also full support for concepts such as labels, atomic prepositions, paths, rewards (transition and state rewards), invariants, race conditions, steady state and transient behaviour, defined in Markov decision processes models (Oxford, 2014).

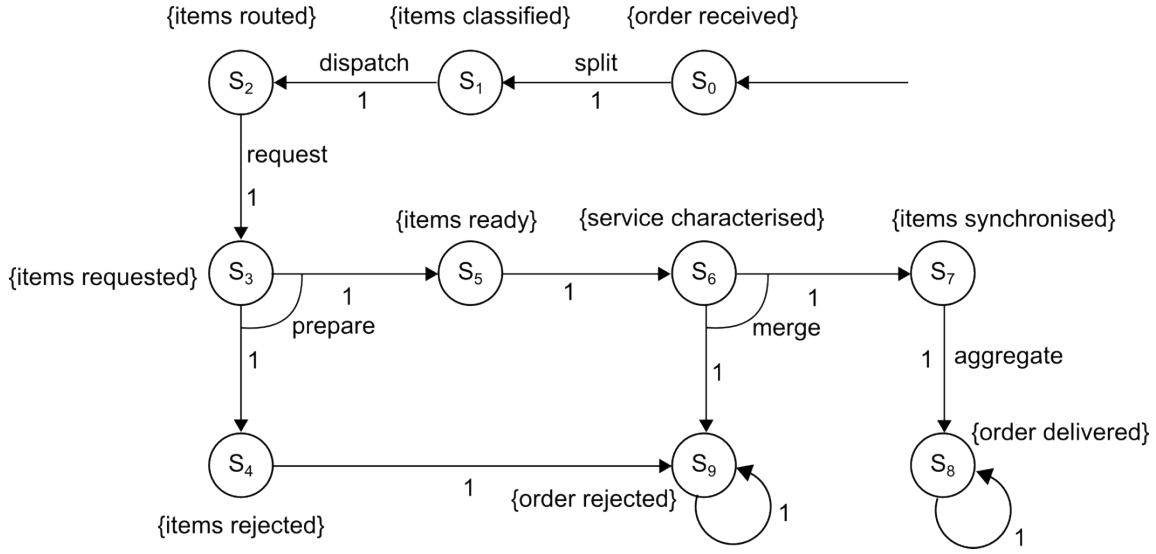

Figure 9: Markov decision process model of a simple (partial) integration solution

Since it is not the purpose of this document to present the full PRISM syntax and functionality, only the simplified formal specification of Figure 9 Markov decision process model is described, as the basis for the PRISM model construction. Assuming a Markov decision process as a tuple M = (S, s_{init}, Steps, L), the example presented in Figure 9 can be described as:

$S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$, the finite set of states, or state space.

$s_{init} = s_0$, the initial state.

AP = {order_received, items_classified, items_routed, items_requested, items_rejected, items_ready, service_chracterised, order_rejected, order_served} , the set of atomic propositions.

L, the labelling with atomic propositions ($L : S \rightarrow 2^{AP}$), which are simply associated to the states in the current example.

$Steps : S \rightarrow 2^{Act \times Dist(S)}$, the transition probability function where Act is a set of actions {split, dispatch, request, prepare, enrich, merge, aggregate} and Dist(S) is the set of discrete probability distributions over the set S. In S all actions were simply associated probability 1, with action "merge" and "prepare" being represented as nondeterministic choices.

Descriptions of computer systems are nowadays possible using several formal techniques. Formal approaches allow for system description (e.g. syntax, grammar, types) and system functional behaviour consistency checking (e.g. deadlocks detection) by the means of calculation or deduction. If formal descriptions are enhanced with expected performance information (e.g. steady-state probabilities, rewards), additional performance measures and properties can be calculated and predicted.

Although Markov processes offer general applicability, they are difficult to construct for large systems, and intermediate description languages are commonly used to overcome this drawback. Queueing Networks and Stochastic Petri Nets are often adopted for this purpose. Queueing networks greatly eases model construction but have limited expressiveness and lack formal interpretation. Stochastic petri nets have a graphical notation and models have formal interpretation, but do not have the explicit structure found in queueing networks. Process algebras have been adopted by researchers in performance analysis due to its advantage of modelling complex systems compositionally and its ability to verify both correct functionality and timeliness of response with the same formalism, preventing consistency issues raised by the usage of different formalisms and techniques for modelling functional behaviour and performance analysis separately (Hillston, 2005).

Performance Evaluation Process Algebra (PEPA) is an algebraic description technique enhanced with time information. It is a stochastic process algebra proposed as a high-level description language for Markov processes. Models specify interactions of components, units or substructures of a system (or roles of its behaviour). They can be atomic or composed of components, perform sets of actions and its behaviour is defined by the the activities in which it can engage. Stochastic process performance measures (both in steady state and transient behaviour) can be extracted from the model, due to some properties of the PEPA formalism and language ensuring a relationship of the process algebra model and a Markov process: It restricts activities duration with the negative exponential distributions, which is the only distribution associate to states duration compatible with the Markov property. The negative

exponential distribution brings the memoryless property to the model. The time until the next event happens is independent from the time of the last event; Also no explicit probabilistic choice operator exists, an implicit choice is assumed by the means of the race condition (Hillston, 2005; Gilmore & Hillston, 1994).

Mitigation of the state space explosion problem raised by most state-based modelling techniques in general and Markov models specifically has been addressed in PEPA. Experience taken from optimisation performed in the context of other formal techniques and exploitation of the PEPA description compositional structure, allows to deal with the state explosion problem while preserving the system model original Markov property. PEPA reasoning engine operates on the derived graph of the PEPA language, which is systematically reduced to a form treatable as a state transition diagram of a stochastic Markovian process. [The PEPA Workbench - workbench.pdf - no abstract] PEPA models have access to the PRISM model checking tool by the means of a compiler that translates PEPA models into PRIMS language (Hillston, 2005).

In order to make PEPA formalisms more appealing to software engineers, a framework for annotated UML models mapping to process algebra descriptions was developed, and used to assess software performance at the design phase. Another similar initiative took place for BPEL4WS models (Hillston, 2005).

FUTURE RESEARCH DIRECTIONS

The innovative perspective introduced in this chapter by the means of integration solutions formal modelling, raised some research questions which will be subject of authors' future work study. Amongst the research topics to be addressed in future work, optimisation of the integration solution formal model is seen as having potentially high impact in integration solutions/systems quality attributes. Choosing parameters of a simulation model optimally may improve overall performance of the simulation system significantly. However, tuning parameters is challenging due to uncertainty that is present in simulation models, which may be caused by either stochastic nature of variables or objective function(s) evaluation (Magoulas et al., 2002).

Stochastic search methods can be used to calibrate parameter values for the simulation model and to find optimal values for resource allocation in integration solutions. Amongst a variety of stochastic search methods, genetic and evolutionary algorithms are considered to be efficient for exploring space of multiple variables simultaneously with one (or several conflicting) objective function(s) to be optimised.

Automatic derivation of a formal model from the integration solution domain-specific language specification is also seen as a theoretical and technical research challenge, targeting the creation of high quality, high productivity and agile EAI software development tools and processes. The information security issues raised by the construction and operation of integration solutions, as highly distributed systems/applications, are also to be addressed in future research. The availability of the integration solution formal model description allows for extensive and accurate exploration of information security attributes checking and analysis.

CONCLUSION

Enterprises are always looking for how to optimise their businesses. Thus, it is not new that software technology plays an important role by providing software applications to support the enterprise's daily activities. All over the years, an enterprise may have purchased or developed in-house several applications, which are commonly not ready to share data and functionality, i.e., they were built without integration concerns in mind. Enterprise Application Integration is a research field that deals with the development of methodologies and tools to design and implement integration solutions. The success of such business processes is highly dependent on the correct and efficient execution of the integration solutions. Simulation can be used to analyse integration solutions aiming at improving their quality before their construction and deployment into production.

There is a high cost, risk, and time spent associated with the simulation approach frequently adopted by software engineers to analyse the behaviour and find possible performance bottlenecks in application integration solutions. It incurs due to the activities related to the construction, execution, and the collection of data from the execution of integration solutions. This chapter has proposed a new approach to cut down cost, risk, and time to deliver better integration solutions. In this approach, integration solutions are seen as discrete-event systems and their conceptual models can be translated into formal models, which are then simulated using well-known techniques and tools for discrete-event simulation. From the state-of-the-art integration technologies available for enterprises to design and implement integration solutions, the domain-specific language of Guaraná was used to illustrate the characterisation of integration solutions as discrete-event systems. Several techniques and tools can be used to support the realisation of a discrete-event simulation. This chapter has explored the use of Queueing theory and Markov based simulation and model checking techniques to analyse conceptual models of integration solutions. These types of probabilistic models have extensive support on modern probabilistic model checking and simulation tools. Thus, a detailed discussion on this kind of models and software tools was also provided.

This chapter addressed two most well-know and relevant techniques in the area of systems formal modelling and analysis, queueing theory based, temporal logic based and process algebra based techniques. These techniques are grounded in strong theoretical foundations, provide textual or/and graphical description languages for systems mathematical modelling and are equipped with tools for simulation and performance analysis. They revealed to be suited to model integration solutions, aiming at the modelling and analysis of slightly different systems properties and perspectives. In one hand, queuing based techniques devote special attention to throughput and resources consumption related properties, in the other hand, temporal logics and probabilistic extensions of temporal logics, are specially focused on model checking, correctness and system constraints related properties, while process algebras based tools presented high potential for systems performance analysis. These tools and analysis perspectives revealed to be of major importance for integration solutions simulation, verification, resources consumption usage prediction and planning in early stages of the integration solutions lifecycle development, especially for the design, implementation, and deployment phases.

Although the state-of-the-art integration technologies have been endowed with a domain-specific language to design integration solutions, the focus of these technologies is still mostly on the implementation and deployment phases. Guaraná is the single technology that provides a platform-independent modelling language and promotes models as first-class citizens. The construction of an

integrated development environment for the EAI domain, including a workbench with a domain-specific language editor, a deployment suite, a runtime system, a monitoring system, and a formal modelling tool for assisted, interactive, optimal integration solutions design, is seen by the authors as an important innovative and fundamental step for the next generation of information systems integration planning, development and exploration processes.

REFERENCES

Aalst, W. M. P. van der (2015). Business Process Simulation Survival Guide, Handbook on Business Process Management, International Handbooks on Information Systems, Jan vom Brocke, and Michael Rosemann, pages 337-370, Springer Berlin Heidelberg.

Aalst, W. M. P. van der, (2010). Business Process Simulation Revisited, Enterprise and Organisational Modeling and Simulation, Lecture Notes in Business Information Processing, Joseph Barjis, Vol (63), pages 1-14, Springer Berlin Heidelberg.

Aalst, W. M. P. van der, Nakatumba, J., Rozinat, A. & Russell, N., (2010). Business Process Simulation: How to get it right?, Handbook on Business Process Management, International Handbooks on Information Systems, J. vom Brocke and M. Rosemann, pages 313-338, Springer Berlin Heidelberg.

Al-Aomar, R. (2010). Simulation Service Systems, In Aitor Goti (Ed.), Discrete Event Simulation (pp. 1-10), Rijeka: Intech.

Babulak, E., & Wang, M. (2010). Discrete Event Simulation: State of the Art, In Aitor Goti (Ed.), Discrete Event Simulation (pp. 141-164), Rijeka: Intech.

Baccelli, F., Cohen, G., Olsder, G. J., & Quadrat, J. P. (1992). Synchronisation and Linearity: An algebra for discrete event systems. Chichester: John Wiley & Sons.

Brémaud, P. (1998). Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues. Texts in Applied Mathematics, NY: Springer.

Cao, X., R., & Ho, Y. C. (1990). Models of discrete event dynamic systems. IEEE Control System Magazine, 10(4), 69-76.

Cassandras, C. G., Lafortune, S. (1999). Introduction to Discrete Event Systems. Kluwer Academic Publishers.

Chwif, L. ; Paul, R. J., & Barreto M. R. P. (2006). Discrete Event Simulation Model Reduction: A Causal Approach. Simulation Practice and Theory, EUA, 14(7), 930-944.

Choi, S.C., & Wette, R. (1969). Maximum Likelihood Estimation of the Parameters of the Gamma Distribution and Their Bias, Technometrics, 11(4), 683–690.

Dossot, D., & D'Emic, J. (2009). Mule in Action. Stamford, CT: Manning.

Desa, W. L. H. M., Kamaruddin, S., Nawawi, M. K. M., & Khalid, R. (2013). Evaluating the performance of a multipart production system using discrete event simulation (DES). International Proceedings of Economics Development and Research, 63(13), 64-67.

DiPerna, R. J., & Lions, P.-L. (1989). On the Cauchy problem for Boltzmann equations: global existence and weak stability. Annals of Mathematics. (2)130, 321-366.

Faget, P., Eriksson, U., & Herrmann, F. (2005). Applying discrete event simulation and an automated bottleneck analysis as an aid to detect running production constraints, Proceedings of the Winter Simulation Conference. Orlando, FL: IEEE.

Fisher, M., Partner, J., Bogoevici, M., & Fuld, I. (2010). Spring Integration in Action. Stamford, CT: Manning.

Forrester, J. W. (1968). Principles of Systems, Cambridge, MA: Wright-Allen Press, Inc.

Frantz, R. Z., & Corchuelo, R. (2012). A Software Development Kit to Implement Integration Solutions. In 27th Symposium On Applied Computing (SAC'12). Riva del Garda, Trento: ACM.

Frantz, R. Z., Reina-Quintero, A. M., & Corchuelo, R. (2011). A Domain-Specific language to design enterprise application integration solutions. International Journal of Cooperative Information Systems, 20(2), 143-176.

Frantz, R. Z., Molina-Jiénez, C., & Corchuelo, R. (2010). On the Design of a Domain Specific Language for Enterprise Application Integration Solutions. In 2nd International Workshop on Model-Driven Service Engineering (MOSE'10). Málaga, Andalucía: CEUR-WS.

Gilmore, S. & Hillston, J. (1994). The PEPA workbench: A tool to support a process algebra-based approach to performance modelling. In Computer Performance Evaluation Modelling Techniques and Tools, Lecture Notes in Computer Science, Günter Haring and Gabriele Kotsis, v. 794, pages 353-368, Springer Berlin Heidelberg.

Gross, D., Shortle, J., Thompson, J., & Harris, C. (2008). Fundamentals of Queueing Theory 4th Edition. New York, NY: Wiley-Interscience.

Haight, F. A. (1967). Handbook of the Poisson Distribution. New York: John Wiley & Sons.

Harrel, C., & Tumay, K. (1994). Simulation made easy: a manager's guide. Atlanta: Institute of Industrial Engineers.

Hilbe, Joseph M. (2011). Negative Binomial Regression, second edition, Cambridge, UK: Cambridge University Press.

Hillston, J. (2005). Process algebras for quantitative analysis. In: Proc. 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005), pages 239-248.

Hlupic, V., (2003). Business Process Modelling: Potential Benefits and Obstacles for wider use. International Journal of Simulation: Systems, Science and Technology, 4(1-2), pages 62-67.

Hlupic, V.; Robinson, S., (1998). Business Process Modelling and Analysis using Discrete-Event Simulation, Simulation Conference Proceedings, Winter, v. 2, pages 1363-1369.

Hohpe, G., & Woolf, B. (2003) Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions. Bostom, MA: Addison-Wesley.

Hohpe, G. (2005). Your coffee shop doesn't use two-phase commit. IEEE Software. 22(2), 64-66.

Hopcroft, J. E., & Ullmann, J. D. (1979). Introduction to Automata Theory, Languages and Computation. Bostom, MA: Addison-Wesley.

Ibsen, C., & Anstey, J. (2010). Camel in Action. Stamford, CT: Manning.

Janssen, M., & Cresswell, A. M. (2005). An enterprise application integration methodology for e-government, Journal of Enterprise Information Management, 18(5), pp. 531-547.

Janssen, J., & Manca, R. (2006). Applied Semi-Markov Processes. New York, NY: Springer.

Jansen-Vullers M. H. & Netjes, M., (2006). Business Process Simulation – a Tool Survey, In Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN.

Jodrá, P. (2012). Computing the Asymptotic Expansion of the Median of the Erlang Distribution. Mathematical Modelling and Analysis 17(2), 281-292.

Kendall, D. G. (1953). Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. The Annals of Mathematical Statistics. 24(3), 338-354.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimisation by Simulated Annealing. Science, 220(4598), 671-680.

Klein, M. J., Sawicki, S., Roos-Frantz F., & Frantz, R. Z. (2014). On the Formalisation of an Application Integration Language using Z Notation. In: Proceedings of 16th International Conference on Enterprise Information Systems, Lisbon: Scitepress.

Kleinrock, L. (1975). Queueing systems, v. 1: Theory, Wiley-Interscience, London.

Kunz, G., Tenbusch, S., Gross, J., & Wehrle, K. (2011). Predicting Runtime Performance Bounds of Expanded Parallel Discrete Event Simulations. In: Proceedings of the IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, Washington, DC: IEEE Computer Society.

Kwiatkowska, M., Norman, G. & Parker, D. (2007). In Stochastic Model Checking. Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation, v. 4486 of LNCS, pages 220-270, Springer.

Kwiatkowska, M., Norman, G. & Parker, D. (2011). PRISM 4.0: Verification of Probabilistic Real-time Systems. In: Proc. 23rd International Conference on Computer Aided Verification (CAV'11), v. 6806 of LNCS, pages 585-591, Springer.

Law, A.M. & Kelton, D.W. (1991). Simulation Modeling and Analysis. New York, NY: McGraw-Hill.

Magoulas, G.A.; Eldabi, T.; & Paul, R.J. (2002). Adaptive stochastic search methods for parameter adaptation of simulation models, Intelligent Systems 2002. In: Proceedings of the First International IEEE Symposium, (2), 23-27.

Oxford University (2014). PRISM Manual version 4.2, Retrieved from http://www.prismmodelchecker.org

Parker, D. (2011). Lectures - Probabilistic Model Checking - Michaelmas Term 2011, Department of Computer Science, University of Oxford.

Paul, R. J. & Balmer, D.W. (1993). Simulation Modelling, Bromley: Chartwell-Bratt Ltd.

Pere, B., Lladó, C. M., Puijaner, R., and Knottenbelt, W. J. (2007). PIPE v2. 5: A Petri Net Tool for Performance Modelling. In Proc. 23rd Latin American Conference on Informatics (CLEI 2007).

Petri, C.A. (1966). Communication with Automata, New York: Griffiss Air Force Base. Tech. Report. RADC-TR-65-377, v. 1, Suppl.1.

Pritsker, A. B. (1986). Introduction to Simulation and Slam. New York, NY: John Wiley & Sons.

Pidd, M. (1998). Computer Simulation in Management Sciences. Chichester: John Wiley & Sons.

Robinson, S. (2004). Simulation: The practice of model development and use. Chichester: John Wiley & Sons.

Rozinat, A., Mans R. S., Song M., & Aalst, W. M. P. van der (2009). Discovering Simulation Models, Information Systems, Vol (34)(3), pages 305-327.

Schmidt, D. F., & Makalic, E. (2009). Universal Models for the Exponential Distribution. IEEE Transactions on Information Theory, 55(7), 3087-3090.

Tohma, Y, Yamano, H., Ohba, M., & Jacoby, R. (1991). The Estimation of Parameters of the Hypergeometric Distribution and its Application to the Software Reliability Growth Model. IEEE Transactions on Software Engineering, 05(17), 483-489.

KEY TERMS AND DEFINITIONS

Integration solution: A piece of software that exogenously orchestrates a set of existing applications, so that they can collaborate by providing data and functionality to support a business process.

Deterministic model: A model which does not have any probabilistic (random) elements. Its output is determined when the set of inputs and relationships in the model have been specified.

Stochastic model: A model that have at least some random input elements. It produces a random output, which must be treated as only an estimate of true characteristics of the system.

Dynamic system: A system that changes its state over time.

Discrete-event system: It is when its state changes only at the instant an event occurs, for all others instants in time, nothing changes.

State of a system: A set of variables used to describe the behaviour of the system at a particular time.

Probabilistic model checking: A formal verification technique for modelling and analysing systems that exhibit probabilistic behaviour.