Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

A TUTORIAL ON ABCmod: AN ACTIVITY BASED DISCRETE EVENT CONCEPTUAL MODELLING FRAMEWORK

Gilbert Arbez
Louis G. Birta

School of Electrical Engineering and Computer Science
University of Ottawa
800 King Edward Ave , P.O. Box 450, Stn A
Ottawa, Ontario, K1N 6N5 CANADA

ABSTRACT

The notion of a conceptual model is present in any discussion about the modelling and simulation process within the discrete event dynamic system domain (Robinson 2011). This paper presents an overview on an activity-based conceptual modelling framework: Activity Based Conceptual modelling = ABCmod (Birta and Arbez 2013). It transforms the general notion of a conceptual model to into a specific conceptual modelling artefact. The ABCmod framework encompasses the naturalness of the activity perspective which has considerable intuitive appeal (Pidd 2004a and 2004b). ABCmod accommodates both the structural and the behavioral aspects that are fundamental components of any conceptual model and provides a collection of constructs both for handling input/output and for dealing with special circumstances such as pre-emption, interruption and balking. We provide an overview of the framework and illustrate many of its features in examples.

1 INTRODUCTION

The development of a meaningful conceptual model is an essential phase for the successful completion of any modelling and simulation project undertaken to solve an identified problem within a system under investigation (SUI). In the realm of continuous time dynamic systems, the conceptual model is a set of differential equations and consequently the language of discourse is the language of this particular domain of mathematics and of the domain from which the underlying dynamic system has emerged (e.g., control, thermodynamics, aerodynamics, etc.). However when the system under investigation falls in the realm of discrete event dynamic systems (DEDS) there is, regrettably, no established language for formulating the conceptual model. The consequence is often a leap directly into the intricacies of some computer programming environment with the unfortunate result that the program displaces the model as the object of discourse. Furthermore, the resulting artefact (i.e., the simulation program) has minimal value if a change in the programming environment becomes necessary.

In this paper we outline a conceptual modelling framework for DEDS which we call ABCmod -- Activity Based Conceptual modelling. The framework provides an informal but disciplined context in which both the structure and the dynamics of the model to be studied can be formulated and discussed in a concise and unambiguous manner. The framework has considerable intuitive appeal because it builds on the familiar notion of entities, events and activities that are fundamental to any dialog about DEDS. Particular care has been taken to ensure that the important notions of input and output are treated in a consistent and coherent way.

The perspective we adopt is that a conceptual model is a carefully constructed artefact that consolidates, in a consistent and coherent manner, those behavioural features of the SUI that are deemed to have relevance to the achievement of project goals. Such a model serves as a descriptive bridge between the generalities of the project description and the precision required for the development of the simulation program that generates the behaviour data acquired over a prescribed observation interval that is required for resolving the project goals. This perspective is reasonably consistent with the notion of conceptual modelling used in Information Systems and Software Engineering as presented in (Mylopoulos 1992).

Apart from the fundamental requirement to capture the essential behavioural features of the SUI, there are two important qualities that any conceptual model within the DEDS domain should have; namely,

a) It must be sufficiently transparent so that all stakeholders in the project can use it as a means for discussing those mechanisms within the SUI that have relevance to the characterization of its behaviour (as interpreted from the perspective of project goals).
b) It must be sufficiently comprehensive so that it can serve as a specification for the computer program that will provide the means for carrying out the simulation study.

The ABCmod framework is informal in nature and is driven by a desire for conceptual clarity. But at the same time, it has a high level of precision, generality and adaptability. To a significant extent, these features flow from the incorporation of software development concepts (looping, decision-making, data structures, etc.). The framework, as presented in the discussions which follow, can accommodate a wide range of project descriptions. However it is not intended to be rigidly defined; it can be easily extended on an *ad hoc* basis when specialized needs arise.

One particular aspect of ABCmod needs to be emphasized; namely, that it must not be interpreted as a programming environment. This is, in fact, reflected in the absence of any discussion about time management. Its purpose instead is to provide an environment for model specification; i.e., a vehicle for making the transition from a project description to a simulation program. The intent is to facilitate the abstraction of the SUI's structure and behaviour without concerns about programming issues and details.

The basic concepts underlying our approach are not new. They can be traced back to the activity scanning paradigm and the three-phase paradigm that are usually identified as one of the modelling and simulation "world views". A comprehensive presentation of activity scanning from a programming perspective can be found in (Kreutzer 1986). Examples of the utilization of this paradigm can be found in (Martinez 2001), (Shi 2000) and (Gershwin 1991). The specific perspective that underlies our approach shares some commonalities with the work of (Overstreet and Nance 2004). The three-phase paradigm is described in (Pidd 2004a).

Note finally that the presentation of the ABCmod framework that is provided in this paper is significantly abbreviated and is intended only to convey its fundamentals. A more complete discussion can be found in Chapters 3 and 4 of (Birta and Arbez 2013).

2 THE LANDSCAPE OF PERTINENT VARIABLES

We begin by noting that sequences of random values, ordered in time, are a fundamental and recurring feature within the DEDS domain. Our particular interest here encompasses a broad spectrum of possibilities; these range from the creation of such sequences to capturing such sequences as flowing from the model. Coupled with the notion of event (changes in the model that occur at discrete points in time), variables in a DEDS model have specific characteristics. This section describes the characteristics adopted in ABCmod and additional variable-related notions that are pertinent to our presentation of the modelling and simulation activity within the DEDS domain. These play an important role throughout the presentation.

2.1 Random Variates, RVV's and RVP's

The execution of a DEDS simulation model can be viewed, in somewhat idealized terms, as a mapping of prescribed random input behavior into random output behavior. The "idealization" at play here arises from the observation that "randomness" within the computing milieu is not, in fact, genuinely random. More specifically, the "random" values that are required in the characterization of random input behavior are, of necessity, samples taken from prescribed probability distributions using algorithmic methods. However, by their fundamental nature, these methods do not guarantee genuine randomness. The values thus generated are called *random variates,* and they are generally regarded as "pseudorandom". In the sequel, we refer to a variable whose values are random variates as a *random variate variable* (RVV).

Within the ABCmod framework, the values for a designated RVV are always provided by a procedure called a random variate procedure that is distinguished with the prefix RVP. The statistical characteristics of the RVV are provided in this procedure. For example, RVP.GetDelta() could be the random variate procedure used to generate values for the random variate variable, delta.

2.2 Discrete Time Variables

A discrete-time variable is a time dependent variable whose value changes only at discrete points in time. Such variables are of fundamental importance in the development of models for discrete event dynamic systems because of their key role in the characterization of both the input and the output of such models.

We recognize two types of discrete-time variable which are called *sample discrete-time variables* and *piecewise constant discrete-time variables* (see Figure 1). The same scheme is used for representing the time-evolution of these two types of variables, namely the characterizing sequence CS[x] which is a sequence of ordered pairs: CS[x] = <(t_k ,x_k): k = 0, 1, 2, ... >. It is convenient to separate the characterizing sequence into two underlying sequences, the *domain sequence for* x, $CS_D[x]$ = <t_k: k = 0, 1, 2 ...>, and the *range sequence for* x, $CS_R[x]$ = <x_k: k = 0, 1, 2 …>.

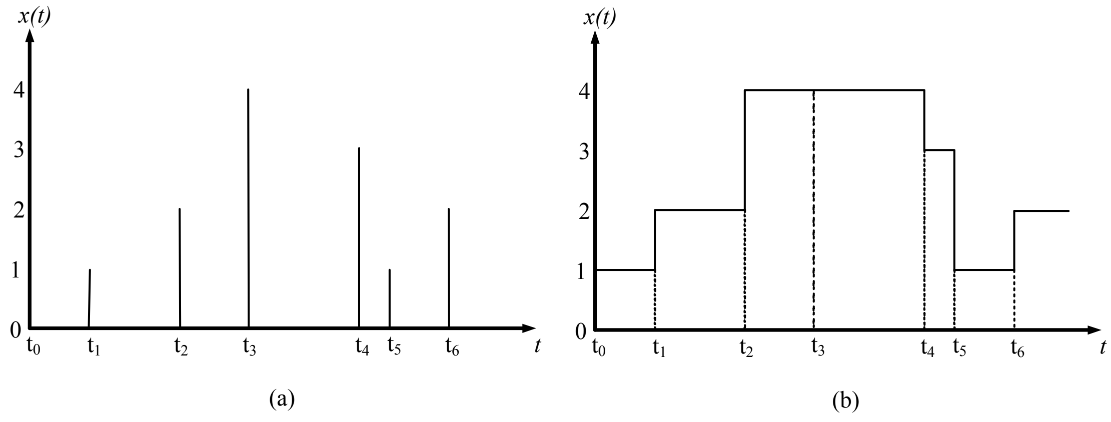


Figure 1: The sample discrete-time variable (a) and the piecewise constant discrete-time variable (b).

The key feature of the sample discrete-time variable $x(t)$ is that its value is meaningful only for those values of t in $CS_D[x]$. On the other hand, a piecewise-constant discrete-time variable, $x(t)$ has a defined value for all values of t within the observation interval. To be specific, our convention will be that $x(t) = x_k$ for $t_k \le t < t_{k+1}$ for $k = 0, 1, 2, \ldots$.

There are, however, differences in manner in which sample discrete–time variables and piecewise constant discrete-time variables are handled in ABCmod. Consequently we shall often refer to the characterizing sequence for a sample discrete-time variable, x, as a *sample sequence* (denoted by PHI[x]) and we shall refer to the characterizing sequence for a piecewise constant discrete-time variable, x, as a *trajectory sequence* (denoted by TRJ[x]).

2.3 Input and Output

Our perspective is that "input" is a facet of the SUI's behavior that has an impact upon it but is not itself affected by that behavior. Not surprisingly, these influences are represented by variables; i.e., input variables. Furthermore the values of input variables are generally associated with a prescribed data model. These data models can have a random basis in which case the input variable is a random variate variable (RVV). The data model can also be deterministic, in which case values are assigned via a deterministic value procedure (DVP) rather than a RVP. We identify two broad categories of input that provide a natural dichotomy of the range of possibilities.

Exogenous input: Generally it can be assumed that there exists a boundary that surrounds the SUI and separates it from its environment. Nevertheless there are almost always aspects of the environment that extend across this boundary and influence the SUI's behaviour. This influence is captured by the *exogenous input* to the SUI. Two types of input variables are associated with exogenous inputs. The *Environmental Input Variables* which are generally piecewise constant discrete-time variables and *Entity Stream Input Variables* are sample discrete-time variables typically used to represent the arrival aspect of a stream of entities that enter the model.

Endogenous input: Often there are facets of the SUI itself that have an impact upon its behaviour but are not themselves influenced by it. This "embedded influence" represents the *endogenous input* to the SUI. Endogenous inputs fall into two categories that we call *independent input variables* and *semi-independent input variables.* In the case of an independent (endogenous) input variable, x, both the domain sequence, $CS_D[x]$, and the range sequence, $CS_R[x]$, are independent of model behaviour. If, on the other hand, x is a semi-independent (endogenous) input variable then only its range sequence, $CS_R[x]$, is independent of model behaviour. Two examples of endogenous input are (i) the varying number of part-time servers working at the counter of a fast food outlet over the course of a day (an independent input variable) and (ii) the service times for the stream of customers that arrive at the counter of that outlet (a semi-independent input variable).

Output is that collection of data generated by a simulation experiment that has relevance to the achievement of the project goals. Consequently it is the fundamental outcome of any simulation experiment. From a modelling perspective output, like input, is identified using suitably defined variables. Because of the stochastic behaviour inherent in the study of DEDS, these output variables are random variables and this has significant implications on how meaningful values are acquired and subsequently interpreted.

A trajectory sequence, TRJ[y], is the means for capturing the values of an output variable, y, that is a piecewise constant discrete-time variable. A sample sequence, PHI[y], is the means for capturing the values of an output variable, y, that is a sample discrete-time variable. Normally the entire collection of data accumulated in a trajectory sequence or a sample set is not of interest. Rather it is some scalar-valued property of the data that has relevance; e.g., average, maximum, minimum etc. The scalar value obtained by carrying out the designated operation at the end of the simulation experiment is assigned to an output variable called a derived scalar output variable (DSOV). The role of a simple scalar output variable (SSOV) parallels that of a DSOV inasmuch as both provide some specific measurable property of either a trajectory sequence or a sample sequence. In the case of the SSOV that property is monitored during the course of a particular experiment and a final value is available at the end of the experiment. This is in contrast to a DSOV whose value flows from a post processing step at the end of an experiment.

3 ABCMOD: THE BASICS

3.1 Fundamental Constituents of the ABCmod Framework

Our perspective is that behavior is the consequence of interaction amongst some collection of objects that populate the space of the SUI. A prerequisite for dealing with this interaction is a means for characterizing these objects. There are, therefore, two collections of modelling artefacts required for the conceptual modelling process. The first deals with the abstraction of the objects that are interacting within the SUI (in effect, the structure of the SUI) and the second focuses on the nature of these interactions (the behavior). These dual requirements are handled in ABCmod with two basic model building artefacts called *entity categories* and *behavioral artifacts.*

The various entity categories that emerge within an ABCmod conceptual model are tailored to the specific nature of the SUI and the goals of the project. Entities are surrogates for the objects of interest within the SUI and each belongs to an identified category. The behavioural artefacts that provide the basis for modelling the SUI's behaviour fall into two categories called activities and actions. These behavioral artifacts effectively encapsulate the SUI specific rules which govern the manipulation of entities

3.2 Model Structure: Entity Categories

An entity is a named *m*-tuple of attributes where we regard an attribute as a discrete-time variable. The values of the attributes of an entity generally vary with time in accordance with the underlying rules of behavior. In reality every entity belongs to a named "category" (of entities); that may have many members; each entity belonging to that category has the same set of attributes.

Every entity category has one or more roles. The role(s) of a category is(are) inherited by all entities belonging to that category. The concept of a role is intended to reflect upon major features that the entities belonging to that category assume in the model. For example, entities with a role of Resource provide a service. Entities with a role of Consumer seek one or more services, usually provided by a resource entity. Entities with a role of Queue serve as the means for maintaining an ordered collection of other entities. Entities with a role of Group serve as a means for maintaining an unordered collection of other entities. Note that roles may be combined. For example a Resource Group role is assigned to a category when its members can service a group of entities at the same time.

Every entity category also has a scope which indicates the number of entities that can exist in that category. The scope Unary indicates that there is exactly one entity in the category and furthermore it is present throughout the observation interval; i.e., it is a permanent member of the SUI. Similarly the scope Set[n] indicates that the entities in the category are permanent, and that the number of entities in the category is fixed at n. Finally the scope Class indicates that the number of entities within the category varies. In other words, the entities belonging to that category have a transitory existence within the SUI

The identification of appropriate attributes for an entity category is governed to a large extent by the requirements that emerge in the process of characterizing dynamic behaviour. The collection of behavioural artefacts used in this characterization within the ABCmod framework, react to and manipulate the attributes of entities. It follows then that the proper selection of the attributes for an entity category is a fundamental step. An attribute may play the role of a system state variable (to reflect the state of the model), an input variable (their value is set from an RVP or DVP) or an output variable (their value is set according to the model behaviour and recorded). An attribute may also be used as a parameter.

Each entity category in an ABCmod conceptual model has a unique name. That name is an integral part of the identifier for individual entities that are members or instances of that category. The identifier for an entity has a format that reflects the properties of the entity category to which it belongs.

3.3 Model Behavior: Activities and Actions

The characterization of behavior in the ABCmod framework is carried out using a collection of behavioral artefacts. These fall into two categories called activities and actions. The notion of an "event" is central to these behavioral artefacts and consequently we begin with an examination of this notion.

Events are, in fact, fundamental to any discussion of model building in the DEDS domain. Within the ABCmod framework we regard an event as an instantaneous change in the status of the model at a point in time (event time). The details of that change are provided by an associated status change specification

(SCS). Note that the changes referred to above do not occur in the conceptual model but rather in the simulation model for which it provides a specification.

Events fall into two broad categories; namely scheduled events and conditional events. A scheduled event is one whose event-time is scheduled (often relative to some other event-time; e.g., use of an RVV to determine the time between the arrivals of two Customers). A conditional event is one whose event-time coincides with the moment when a precondition (a Boolean condition formulated on one or more state and/or input variables) of the model becomes TRUE. There are, in addition, two variations; namely, a tentatively scheduled event (the first variation) which is a scheduled event whose occurrence may be stopped by an intercepting event (the second variation which is a conditional event) should it occur before the event-time of the tentatively scheduled event. The latter two events are relevant when developing activities that may be "interrupted".

The ABCmod activity is the main modelling artefact for characterizing behaviour. An ABCmod conceptual model typically incorporates a number of activities. Each activity has a name and serves to represent a unit of behaviour that has been identified as having relevance from the perspective of the project goals. The notion of "unit" here is intended to suggest "minimality"; in other words, an activity should be viewed as being "atomic" in the sense that it captures an aspect of the model's specified behavior that is not amenable to subdivision (at least from the perspective taken by the model builder). An activity can also be regarded as an abstraction of some purposeful task that takes place within the SUI. The key consequence of both the initiation and the completion of this task generally take the form of changes in the value of some of the state variables within the model; i.e., attributes of entities. In summary, there are three noteworthy features of an ABCmod activity construct: it represents an indivisible unit of behaviour that occurs within the SUI, it is associated with some purposeful task, and it evolves over a nonzero interval of time (its duration).

The ABCmod activity incorporates, at least two event specifications; namely, one for a starting event and one for a terminating event. Note that the specification of a starting event may include multiple event-times each of which creates an instance of the activity. An instance of an activity exists between the event-time of its starting event and the event-time of its terminating event. In the case where there are exactly two events, the terminating event is always a scheduled event whose event time is dependent on the duration specified for the activity.

The starting event of an activity may be either a scheduled event or a conditional event. The first case gives rise to a scheduled activity which contains the following components: a time sequence (often provided by an RVP) which specifies a sequence of event-times of the starting event, the status change specification (SCS) of the starting event, the duration which when added to the event-time of the starting event gives the event-time of the terminating event, and the SCS of the terminating event which specifies the changes that occur in the model at the event-time of the terminating event. The conditional activity has the same components as the scheduled activity except that the initiation of the activity is governed by a precondition rather than a time sequence.

It is common to have an activity instance immediately follow the ending of another activity instance, that is, the event-time of the starting event of the second activity instance coincides with the event- time of the terminating event of the first activity instance. The second activity in this sequence is called a sequel activity. A sequel activity has the same components as a scheduled activity except that the time sequence is replaced by a parameter list that serves to transfer information into the sequel activity instance.

When an activity (any of three types of activities outlined above) can be interrupted, it is augmented with additional events; furthermore, the terminating event becomes a tentatively scheduled event. The events added to the activity are intercepting events each of which has a precondition and an SCS. Such activities are referred to as extended activities.

The action is the second category of behavior modelling artefact within the ABCmod framework. While activities serve to capture the various relevant tasks that are carried out within the SUI, an action simply provides the means for characterizing events that are not embedded within activities. Because an action is simply an event it unfolds at a single point in time and consequently the concept of "instances" is not meaningful. However multiple occurrences of the action are typical. There are two types of actions and they are called the scheduled action and the conditional action. Their difference parallels the difference between the scheduled activity and the conditional activity as outlined earlier.

4 THE ABCmod FRAMEWORK

Dealing with complexity in any context is considerably facilitated when it can be addressed at more than one level of detail. This view has, in fact, been adopted in the ABCmod framework inasmuch as conceptual model development evolves as a two stage process. We refer to the initial stage as "high level" inasmuch as its purpose is restricted to providing a preliminary perspective that is unencumbered by excessive detail. This version of the model is primarily intended for discussion amongst all project stakeholders. The detail is introduced at a second stage of development which is called the "detailed level". This version of the model is a detailed specification intended for the simulation modelling team which will translate the conceptual model into a simulation program.

The framework is introduced here by presenting a simple example called Kojo's Kitchen. Kojo's Kitchen is one of the fast food outlets in the food court of a shopping mall. Kojo's manager has been receiving complaints from customers about long waiting times. He is interested in exploring staffing options to reduce these complaints. The mall (and hence Kojo's) is open between 10:00 am and 9:00 pm every day. Kojo's serves only two types of product; namely, sandwiches and sushi. Two rush hour periods occur during the business day, one between 11:30 am and 1:30 pm, and the other between 5:00 pm and 7:00 pm. Currently two employees work at the counter throughout the day serving customers with freshly prepared sandwiches and sushi products.

4.1 Project Goals: Parameters, Experimentation and Output

Often experimentation is concerned with exploring changes in behavior that result from various values assigned to defined parameters. Parameters can emerge in either of two contexts. The first context relates to the model itself and the second to the model input. In the first case, a parameter can alter either a structural aspect of the model or its rules of behavior. In addition it is necessary to identify the output needed to solve the problem. Figure 2 summarizes this view for Kojo's Kitchen.

Note that the parameter, addEmp, is an attribute of the entity category called Counter. The attribute name is "RG.Counter.addEmp" according to the ABCmod standard convention for naming attributes. It consists of a prefix RG (for Resource Group) that identifies the roles associated to the entity category whose name is Counter.

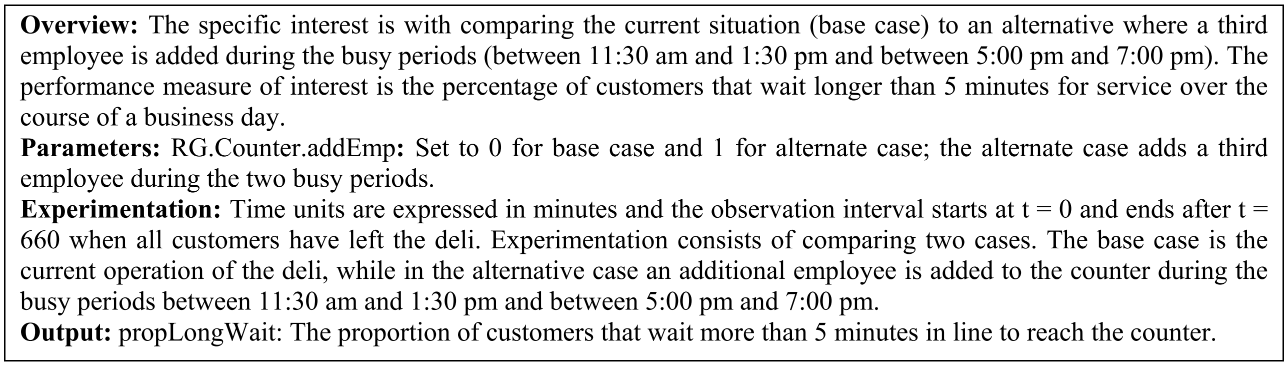

Overview: The specific interest is with comparing the current situation (base case) to an alternative where a third employee is added during the busy periods (between 11:30 am and 1:30 pm and between 5:00 pm and 7:00 pm). The performance measure of interest is the percentage of customers that wait longer than 5 minutes for service over the course of a business day.
Parameters: RG.Counter.addEmp**:** Set to 0 for base case and 1 for alternate case; the alternate case adds a third employee during the two busy periods.
Experimentation: Time units are expressed in minutes and the observation interval starts at t = 0 and ends after t = 660 when all customers have left the deli. Experimentation consists of comparing two cases. The base case is the current operation of the deli, while in the alternative case an additional employee is added to the counter during the busy periods between 11:30 am and 1:30 pm and between 5:00 pm and 7:00 pm.
Output: propLongWait: The proportion of customers that wait more than 5 minutes in line to reach the counter.

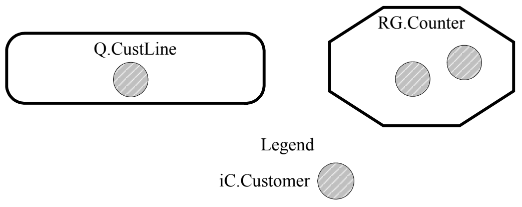
Figure 2: Kojo's Kitchen project goals.

4.2 High Level ABCmod Conceptual Model

The High Level ABCmod conceptual model provides the means for discussing the model among all stakeholders. It consists of four parts: simplifications and assumptions, a structural view, a behavioral view and input.

In Kojo's Kitchen, we adopt the following simplification: there are only two types of customer; one type purchases only sandwiches and the other type purchases only sushi products. Discussion of simplification and assumptions can be found in (Robinson 2008) and (Birta and Arbez 2013).

The purpose of the structural view is to identify the entity categories. The structural view consists of a structural diagram followed by a description of each entity category. Such descriptions generally do not include the description of attributes. The structural view for Kojo's Kitchen is shown in Figure 3. The shape of the icons used in the structural diagram for the different entities and/or entity categories reflects their respective roles.



Structural Diagram

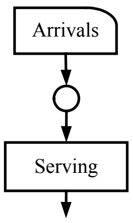
- **CustLine**: An entity category with role = Queue (denoted by the prefix Q) and scope=Unary that represents the line of customers in front of the counter.
- **Counter**: An entity category with role = Resource Group and scope = Unary that represents the counter where customers are being served.
- **Customer**: An entity category with role=Consumer and scope = Class (the "i" in the prefix denotes that the entity category has scope = Class) that represents the collection of customers requiring service at the counter. There are two types of customers, the sandwich customer and the sushi customer (distinguished using the attribute *uType*).

Figure 3: Kojo's Kitchen structural view.

The behavioral view identifies the set of Activities (and Actions) for the model. The behavioural view consists of a behavioural diagram followed by a description of each action and activity. The behavioural diagram is a collection of life-cycle diagrams which show how an entity can transition from one activity instance to another. It therefore provides a means for identifying all possible activity sequences that an entity can follow. The behavioral view for Kojo's Kitchen is shown in Figure 4 (it consists of a single life-cycle for the Customer entity).

In general, a life-cycle diagram has many segments where each segment shows the transition from one activity instance to another. Activity instances in the life cycle diagram are represented by labelled rectangles. Actions on the other hand have a rounded corner. In the most typical case an entity's transition from one activity instance to another encounters a delay. This delay can occur for a variety of reasons (usually because of inhibiting preconditions) and we indicate this possible delay with a circle which we call a *wait point*. The wait point simply shows that the entity's transition to a subsequent activity instance is not necessarily immediate and that the subsequent activity is a conditional activity. Furthermore, it is possible for several arrows to emanate from the wait point which shows that the entity may transition to any one of several activity instances. The transition will ultimately be to the one whose precondition first becomes TRUE.

Table 1 shows the specification of input for Kojo's Kitchen. The domain sequence of the input entity stream variable uC provides the arrival times of customers and it is associated with the action Arrivals (see section 4.3.2.2) to completely specify an input entity stream.



Behavioural Diagram

Actions:

- **Arrivals**: This scheduled action generates the input entity stream of customers.
- **StaffChange**: Generates the change in the number of employees behind the counter. This Action is not shown in the behavioural diagram since it is not part of the customer life cycle.

Activities:

- **Serving**: This activity represents the serving of a customer.

Figure 4: Kojo's Kitchen behavioral view.

The attribute RG.Counter.uNumEmp attribute is an input variable; the domain and range sequence for this input variable is explicitly specified in Table 1 (note the use of the parameter RG.Counter.addEmp). The two semi-independent input variables serve to initialize the attribute iC.Customer.uType when a customer arrives (see the action Arrivals in Section 4.3.2.2) and to define the service time for the Serving activity (see Section 4.3.2.3).

Table 1: Kojo's Kitchen input variables.

Exogenous Input (Entity Stream)			
Variable Name	**Description**	**Domain Sequence**	**Range Sequence**
uC	This input entity stream represents the arriving customers	RVP.DuC()	1 customer arrives at each arrival.
Endogenous Input (Independent)			
Variable Name	**Description**	**Domain Sequence**	**Range Sequence**
RG.Counter.uNumEmp	Represents the number of employees at the counter.	< 0, 90, 210, 420, 540>	< 2, 2 +RG.Counter.addEmp, 2, 2 +RG.Counter.addEmp, 2 >
Endogenous Input (Semi-independent)			
Variable Name	**Description**		**Value(s)**
iC.Customer.uType	Provides the type of an arriving customer.		RVP.uType()
uSrvTm	Service time of a customer.		RVP.uSrvTm()

4.3 Detailed ABCmod Conceptual Model

The distinction between the structural and the behavioural components of an ABCmod conceptual model is retained in the detailed level presentation. The discussion which follows is accordingly separated. Section 4.3.1 outlines the two structural components of an ABCmod conceptual model; it is a table that provides the constants and parameters (Table 2) and a set of tables that specify each entity category. Section 4.3.2 presents the collection of behavioural components. It begins with details relating to initialization which is followed by a set of tables that specify output, input constructs and the behavioural constructs (actions and activities).

4.3.1 Structural Components

Because there are no constants defined for the Kojo's Kitchen model Table 2 lists only parameters. Tables 3 to 5 provide the specifications for the three entity categories of the model.

Table 2: Kojo's Kitchen parameters.

Parameters		
Name	**Description**	**Value**
RG.Counter.addEmp	Number of additional employees serving at the counter during the busy periods.	0 (for the base case) or 1 (for the alternate scenario).

Table 3: An entity category with role Consumer and scope Class.

Consumer Class: Customer	
This entity structure represents the customers who arrive at Kojo's Kitchen to make a purchase.	
Attributes	**Description**
startWaitTime	Time stamp that holds the value of time when the customer enters the customer line.
uType	The assigned value indicates the type of customer. Two values are possible: 'W' for a sandwich customer or 'U' for a sushi customer.

Table 4: An entity category with a single member and dual roles of Resource and Group.

Resource Group Unary: Counter	
This resource group represents the counter where customers receive service.	
Attributes	**Description**
list	The group of the Customer entities that are being served.
n	Number of entries in list (necessarily $n \leq$ uNumEmp).
uNumEmp	Input variable that provides the number of employees.
addEmp	A parameter whose value is the number of additional employees serving at the counter during the busy periods.

Table 5: An entity category with role Queue and a single member.

Queue Unary: CustLine	
The queue of customers waiting for service at the counter.	
Attributes	**Description**
list	List of the customers entities waiting in line for service. Discipline: FIFO.
n	The number of iC.Customer entities in list.

4.3.2 Behavioral components

This section begins with the specification of the observation interval (time units and range) as shown in Figure 5 followed by an initialization action (with a single starting event time) as shown in Table 6. In the case of Kojo's Kitchen, the right hand side of the observation interval consists of an implicit stop condition which means that the interval ends on a condition rather than a specific time.

Time units:	minutes
Observation interval:	$t_0 = 0$ (10:00am) , Implicit Stop Condition: ($t_f > 660$ minutes (9:00 pm) AND RG.Counter.n = 0).

Figure 5: Specification of the observation interval.

Table 6: An action that occurs at the start of the observation interval to provide initialization (the "SSOV" prefix indicates that the named variable is a simple scalar output variable).

Action: Initialise	
TimeSequence	< 0 >
Event SCS	RG.Counter.n ← 0; Q.CustLine.n ← 0; SSOV.numServed ← 0; SSOV.numLongWait ← 0

4.3.2.1 Output

Table 7 provides the specification of three SSOV's. The numServed and numLongWait variables are used to derive the value of the propLongWait output variable identified in the project goals.

Table 7: The three SSOV's for Kojo's Kitchen.

OUTPUTS	
Simple Scalar Output Variables (SSOV's)	
Name	**Description**
numServed	Number of customers served.
numLongWait	Number of customers that waited longer than 5 minutes.
propLongWait	numLongWait/numServed.

4.3.2.2 Input Constructs

This section may contain random variate procedures (RVP), deterministic value procedures (DVP) and user defined procedures (UDP) as well as actions and activities all related to specifying input. The RVPs shown in Table 8 illustrate typical uses for RVPs, that is, to define arrival times, to define values for attributes, and durations for activities.

Table 8: Random variate procedures.

Random Variate Procedures		
Name	**Description**	**Data Model**
RVP.DuC()	Provides the values of the arrival times of customers. No arrivals are allowed after closing (i.e. for $t \geq 660$).	Exponential(X) where X is: MEAN1=10, for $0 \leq t < 90$ MEAN2=1.75, for $90 \leq t < 210$ MEAN3=9, for $210 \leq t < 420$ MEAN4=2.25, for $420 \leq t < 540$ MEAN5=6, for $540 \leq t < 660$ (All values are in minutes.)
RVP.uType()	Provides the type of the arriving customer. Returns either 'W' (sandwich customer) or 'U' (sushi customer).	Proportion of sandwich customers: PROPW=65% Proportion of sushi customers: PROPU=35%
RVP.uSrvTm(type)	Provides a value for the service time of customer according to the value of type.	For type 'W' (sandwich customer): UNIFORM(STWMIN, STWMAX) where STWMIN=3 min. and STWMAX=5 min. For type 'U' (sushi customer): UNIFORM(STUMIN, STUMAX) where STUMIN=5 min. and STUMAX=8 min.

Table 9 shows the action associated with the input entity stream specification. The domain sequence of the input variable uC determines the arrival times of customers and the action's event SCS derives the instance of the arriving entity, initializes its attributes and inserts it into a queue. The SP prefix denotes a standard procedures pre-defined in the ABCmod framework. For example SP.Derive(Customer) is the standard procedure that generates an instance of the Customer entity category. The variable t is the current time, i.e., the event-time of the action. Table 10 specifies an action used to update an input variable.

Table 9: This action construct is part of the input entity stream specification for Customer arrivals.

Action: Arrivals	
The input entity stream of arriving customers.	
TimeSequence	RVP.DuC()

Event SCS	iC.Customer ← SP.Derive(Customer)
	iC.Customer.uType ← RVP.uType()
	iC.Customer.startWaitTime ← t
	SP.InsertQue(Q.CustLine, iC.Customer)

Table 10: An action that serves to update the value of an input variable.

Action: StaffChange	
Manages the value of the input variable RG.Counter.uNumEmp, that is, schedules an additional employee during busy times for the alternate case.	
TimeSequence	<0, 90, 210, 420, 540 >

Event SCS	IF(t=0 OR t= 210 OR t=540) THEN RG.Counter.uNumEmp ← 2
	ELSE IF(t=90 OR t = 420) THEN RG.Counter.uNumEmp ← 2 + RG.Counter.addEmp
	ENDIF

4.3.2.3 Behavioral Constructs

The conditional activity shown in Table 11 captures the behavior associated with serving customers. Up to three instances of this activity can coexist, that is, up to RG.Counter.uNumEmp customers may be serviced at the same time. Note the portion of the SCS that updates the SSOV's. The standard procedure SP.InsertGroup() by definition adds the customer to the Counter resource group and updates RG.Counter.n.

Table 11: A conditional activity.

Activity: Serving	
Service for a customer.	
Precondition	(RG.Counter.n < RG.Counter.uNumEmp) AND (Q.CustLine.n ≠ 0)

Event SCS	iC.Customer ← SP.RemoveQue(Q.CustLine)
	IF(t – iC.Customer.StartWaitTime > 5) THEN
	    SSOV.numLongWait +← 1
	ENDIF
	SSOV.numServed +← 1
	SSOV.propLongWait ← SSOV.numLongWait/SSOV.numServed
	SP.InsertGroup(RG.Counter, iC.Customer)

Duration	RVP.uSrvTm(iC.Customer.uType)

Event SCS	SP.RemoveGroup(RG.Counter, iC.Customer)
	SP.Leave(iC.Customer)

5 DEALING WITH INTERRUPTIONS

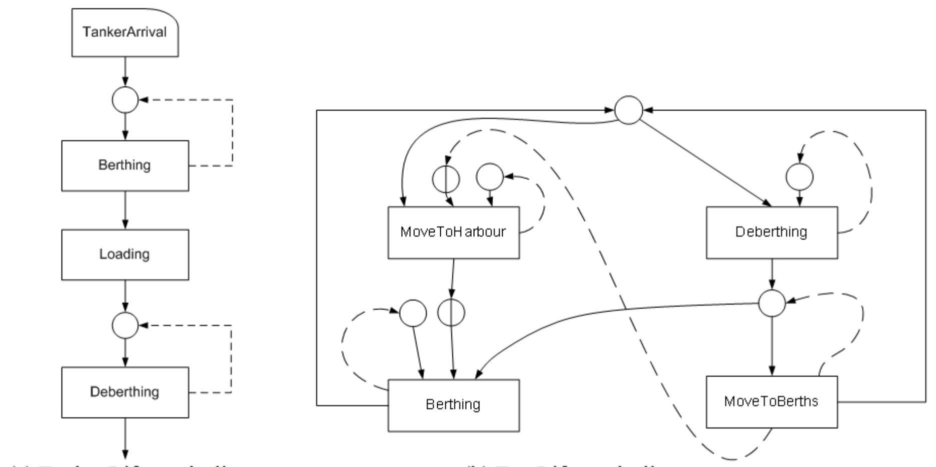
An important feature incorporated into the ABCmod framework is a means for dealing with interruptions. As described in Section 3.3, the mechanism used is an extended activity that contains at least 3 events, the starting event, the terminating event (which is a tentatively scheduled event) and an intercepting event.

We use a port example to illustrate this feature. In this example, a single tug moves empty tankers from the harbor to one of 3 berths (berthing) for loading with oil. The tug will also move the filled tankers back to the harbor once tankers are filled (deberthing). The tug may also move between the harbor and berths in both directions with no tanker in tow. The objective of the simulation study is to evaluate the impact of a fourth berth on tanker waiting times.

Two types of interruptions can occur. The first is when a storm arrives in the port. If the tug is moving between the harbor and the berth, it must stop, drop anchor and resume its activity once the storm has passed. The second is when the tug is moving back from an empty harbor to the berths with no tanker in tow (to be ready for deberthing a tanker being loaded); if a tanker arrives in the empty harbor, and the tug has not completed 70% of its journey back to the berths or there are no tankers waiting to be deberthed, the tug will return to the harbor entrance to service the newly arrived tanker.

Figure 6 illustrates how interruptions are shown in life-cycle diagrams using dashed lines that represent the occurrence of an interruption (the precondition of an intercepting event becomes TRUE). When the input variable uStorm (see Table 12) assumes the value of TRUE, any activity involving the tug (Berthing, Deberthing, MoveToHarbour, MoveToBerths) is suspended (i.e., interrupted, see

SP.Terminate() in Table 12). When the storm is over (uStorm set to FALSE) the interrupted activity is re-instantiated to complete its task.



(a) Tanker Life-cycle diagram (b) Tug Life-cycle diagram

Figure 6: Port behavior diagram.

In the case of the MoveToBerths activity, the activity may also be interrupted by the arrival of a tanker into the empty harbor as previously described. This is shown by the second dashed arrow leaving the MoveToBerths activity to the null wait point (wait point with the line across it) which indicates that the precondition to the MoveToHarbour activity is known to be TRUE.

Table 12: An example of an extended conditional activity.

Activity: MoveToHarbour	
This activity represents the Tug entity moving to the harbour area with no Tanker entity in tow. This activity can be initiated after an interruption: either an interruption of the MoveToBerths activity (because a tanker has arrived while moving to the berth area without a tanker in tow) or an interruption of the MoveToHarbour activity caused by a storm.	
Precondition	(uStorm = FALSE) AND (((R.Tug.status = PAUSE_B) AND (Q.DeberthingList.n = 0) AND (Q.BerthingList.n > 0) AND (RG.Berths.n < RG.Berths.numBerths)) OR ((R.Tug.anchored = TRUE) AND (R.Tug.Status = TO_HARBOUR)) OR (R.Tug.returnToHarbour = TRUE))

Event SCS	IF(R.Tug.status = PAUSE_B) ***(Normal conditions)***
	R.Tug.status ← TO_HARBOUR
	R.Tug.travelTime ← EMPTY_TIME
	ELSE IF(R.Tug.anchored = TRUE) R.Tug.anchored = FALSE ***(After storm)***
	ELSE R.Tug.returnToHarbour = FALSE ***(Tanker has arrived in harbor)***
	ENDIF
	R.Tug.startTime ← t

Duration	R.Tug.travelTime
Interruption Precondition	uStorm = TRUE

Event SCS	R.Tug.travelTime - ← t - R.Tug.startTime
	R.Tug.anchored ← TRUE
	SP.Terminate()

Event SCS	R.Tug.status ← PAUSE_H

The starting event precondition of the extended activity MoveToHarbour as given in Table 12 contains three parts which can only be applied if there no storm is present (uStorm = FALSE). The first part of the precondition is TRUE when (1) the tug is at the berths (R.Tug.status = PAUSE_B), (2) there are no tankers to be deberthed (Q.DeberthingList.n = 0), (3) there are tankers in the harbor waiting to be berthed (Q.BerthingList.n > 0) and (4) there is room at the berths to receive the tanker (RG.Berths.n < RG.numBerths). This part initiates the activity under normal conditions. The other two parts of the starting event precondition accommodate the two interruptions. The part ((R.Tug.anchored = TRUE) AND (R.Tug.Status = TO_HARBOUR)) indicates that the tug was interrupted by the storm while moving to the harbor while the part (R.Tug.returnToHarbour = TRUE) indicates that the MoveToBerths activity was interrupted by a tanker arriving at the harbor. The starting event SCS also evaluates these conditions and takes appropriate actions.

Intercepting events are specified below the Duration specification of the activity as shown in Table 12. The dash lines separating the Duration field from the precondition and event SCS fields of the intercepting event emphasizes that the event can occur during the duration interval of the activity. The event SCS of the intercepting event will indicate that anchor is dropped by setting the R.Tug.anchored to TRUE and computes and saves in the attribute R.Tug.travelTime the time left to travel.

6 CONCLUSION

The overview of the ABCmod conceptual modelling framework for discrete event dynamic systems presented in this paper will hopefully provide some motivation for the adoption of this activity-based approach to conceptual modelling within the DEDS domain. The authors would like to emphasize that page limitations have constrained the presentation. Consequently there are numerous facets of the approach which have been presented in only a rudimentary/superficial manner. Included here are: the use of entity categories with scope=Set[n] for dealing with a fixed but a multiple number of members (namely, n) of an entity category, the use of user-defined procedures for handling complexity, the collection of standard procedures that are available to carry out frequently required operations (e.g., insertion (removal) of entities into (out of) queues or groups), iconic conventions for representing the various types of entity categories, the application of scheduled activities and sequel activities, etc. These features all considerably enhance the ease with which the approach can be applied in accommodating the substantial complexities that can arise in conceptual model development within the DEDS landscape. Elaboration and illustration of these features can be found in (Birta and Arbez 2013).

It should also be pointed out that although ABCmod was developed as a conceptual modelling environment, it has been recognized that the ultimate requirement is for "executable code" namely a simulation model. Recent work has, in fact, produced a new world view, the Activity Object World View (Arbez and Birta 2010), which provided the basis for the development of a program library called ABSmod/J for transforming an ABCmod conceptual model into a Java program. The main feature of this new world view is that it preserves the modelling artefacts of ABCmod (both entity categories and activities/actions) as objects in the simulation program. In effect, then, a comprehensive environment for both conceptual model development and simulation experiments has been developed. The ABSmod/J tool is available on request from the first author.

REFERENCES

Arbez, G., and L. G. Birta. 2010. "An activity-object world view for ABCmod conceptual Models." In *Proceedings of the 2010 Summer Computer Simulation Conference,* edited by Wainer G., 290-297. San Diego, CA: Society for Computer Simulation International.

Birta, L. G., and G. Arbez. 2013. *Modelling and Simulation: Exploring Dynamic System Behaviour*. 2nd ed. London, England: Springer-Verlag.

Gershwin, S. B. 1991. "Hierarchical Flow Control: A Framework for Scheduling and Planning Discrete Events in Manufacturing Systems." In *Proceedings of the IEEE, Special Issue on Dynamics of Discrete Event Systems* 77(1): 195-209.

Kreutzer, W. 1986. *System Simulation: Programming Styles and Languages*. Boston, MA, USA, Addison-Wesley Longman Publishing Co, Inc.

Martinez, J. C. 2001, "EZStrobe: General-Purpose Simulation System Based on Activity Cycle Diagrams" In *Proceedings of the 2001 Winter Simulation Conference*, edited by B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 1556-1564. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Mylopoulos, J. 1992. "Conceptual Modeling and TELOS." In *Conceptual Modeling, Databases, and Case: An Integrated View of Information Systems Development,* edited by Loucopoulos P. and Zicari R., Chapter 2. New York, NY: John Wiley & Sons

Overstreet, C. M., and R. E. Nance. 2004. "Characterizations and Relationships of World Views in Proceedings" *Proceedings of the 2004 Winter Simulation Conference*, edited by R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 279-287. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pidd, M. 2004a, *Computer Simulation in Management Science*, 4th ed. Chichester, England: John Wiley & Sons, Inc.

Pidd, M. 2004b. "Simulation World Views – So What?" In *Proceedings of the 2004 Winter Simulation Conference*, edited by R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 288-292. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Shi, J. J. 2000. "Object-Oriented Technology for Enhancing Activity-Based Modelling Functionality." In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang and P. A. Fishwick, 1938-1944. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Robinson, S. 2008. "Conceptual Modelling for Simulation Part I: Definition and Requirements." *Journal of the Operational Research Society* 59: 278-290.

Robinson, S. 2011. "Choosing the Right Model: Conceptual Modeling for Simulation." In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. Creasey, J. Himmelspach, K. P. White, and M. Fu, 1423-1435. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

GILBERT ARBEZ is an Assistant Professor at the School of Electrical Engineering and Computer Science (EECS). He has been teaching university courses since 1994 and became a full time teacher at EECS in 2003. Part of his teaching load includes a course on computer modelling and simulation fundamentals. He has co-authored a book on Modelling and Simulation fundamentals which features a conceptual modeling framework; he has subsequently contributed to a chapter in a conceptual modeling book and contributed a number of papers on the topic. Current interests include modelling and simulation of discrete event dynamic systems and continuous systems as well as teaching in engineering. His e-mail is garbez@uottawa.ca.

LOUIS BIRTA is an Emeritus Professor in the School of Electrical Engineering and Computer Science at the University of Ottawa. His research career has focused on a broad range of topics which include parallel methods for continuous system simulation, optimization, validation of simulation models and conceptual modeling. He was, for several years, a member of the Board of the Society for Modeling and Simulation (SCS), has served as editor for the SCS journal SIMULATION, and was founding editor of the SCS M&S Magazine. He is a co-author (with Gilbert Arbez) of an introductory book (second edition) on modeling and simulation and is currently Series Editor of the recently initiated Springer book series called *Simulation Foundations, Methods and Applications*. His email address is: LBirta@uottawa.ca.