Preliminary Investigations on Stochastic Information Control Nets

Authors:
Clarence (Skip) Ellis, University of Colorado, USA
Kwang-Hoon Kim, Kyonggi University, Korea
Aubrey Rembert, IBM Corporation, USA
Jacques Wainer, State University of Campinas, Brazil

Department of Computer Science
University of Colorado at Boulder
Technical Report CU-CS-1055-09

Abstract

In this document, we extend the notions of classic Information Control Nets [8] to define new concepts of Stochastic Information Control Nets. We introduce a simple and useful AND-probability semantic and show how this probabilistic mathematical model can be used to generate probabilistic languages. As our introduction, we define and explain the concepts of the Information Control Nets framework. The notion of a probabilistic language is introduced as a normalizer for comparisons. We discuss model-log conformance. We describe a recursive function and an algorithm for generation of probabilistic languages from stochastic ICNs. An important aspect of our generation algorithm is that it generates probabilistic languages that are normalized, and in probability ranked order. We present new augmented definitions of completeness, correctness, and fidelity of a model. These definitions can be applied to many aspects of organizational modeling including the process, the informational, and the resource perspectives. The model that we introduce here can be used to augment and expand on analyses that have been useful and insightful within varied organizational analysis and organizational engineering applications.

1. Introduction

The design, analysis, development, and successful evolution of today's complex organizations require models and methodologies that capture and integrate the many aspects of a multi-dimensional organization. The Information Control Net (ICN) is an open-ended meta-model conceived over 30 years ago as a family of models for organizational understanding, description, analyses, and implementation [7, 8]. The ICN consists of perspectives that include the process, the informational, and the resource perspectives. It allows organizational analysis that transcends the single perspective limits in a powerful and uniform manner [10]. In this document, we introduce Stochastic Information Control Nets (s-ICNs) as a probabilistic extension of ICNs. This allows us to do both qualitative and quantitative organizational engineering, as illustrated later in this document. The main thrust of this document is concerned with presentation of recursive functions and algorithms which generate normalized probabilistic languages from Stochastic Information Control Nets.

S-ICNs enable organizational *model – log checking*. Organizations have multiple models – some may be explicit and formal; others are implicit and informal. Organizations have multiple historical information repositories [22]. For example, some may be transaction logs, communication records, audit trails, etc. By noting the relative frequency of various event types in a log (or repository), we can assign occurrence probabilities to these event types. By assigning probabilities to the AND / OR forks in a process model, we can calculate the probability of every execution sequence of the model. Then we can compare these two probability distributions. This calculation and comparison can be complex and non-obvious when the model has loops (infinite sequences), and the log is quite large with anomalous entries [18]. If this comparison is successful, then we can define metrics such as the stochastic model-log conformance of a model with respect to a log to quantitatively gauge our conceptual grasp of the organization. Concepts and examples of the s-ICN model, the nature of the log, and the probabilistic language generation algorithm for model-log comparison are explained and motivated in more detail in the remainder of this document.

In section 2, we present the ICN Meta-Model. Section 3 introduces our **Stochastic Information Control Nets.** Section 4 defines the concept of a Probabilistic Language (p-Language). An anomalous example is presented of an s-ICN which generates a non-normalized p-Language. In section 5, we discuss model-log comparison; followed by a summary of related work in section 6. Our algorithm to generate p-Languages from s-ICNs is presented in 7. This is extended in sections 8 to cover s-ICNs with loops. P-Languages generated by s-ICNs with loops are infinite, so we discuss maximal string generation, and approximation language generation in section 9. In section 10, we present summary and conclusions; acknowledgements in 11, and bibliography in 12.

2. The ICN Meta-Model

Different organizations have different goals, different structures, and different organizational styles. Therefore, different organizations typically need different methodologies, different workflow products, and different models to express different business perspectives. The concept of a meta-model provides a coherent, uniform notation and a set of conceptual tools for creating various models appropriate to various organizations. The Information Control Net Meta-Model represents a family of models that have been designed and used to model various aspects of organizations and business process management.

The ICN modeler chooses certain objects of interest and structures from an organizational framework, from an organizational schema, and from an organizational net. The organizational framework is used to specify various classes of organizational objects (e.g. goals, constraints, resources, activities). The organizational schema is used to specify the set of mappings over the classes of organizational objects (e.g. who does what, which activities precede which). The organizational net is used to specify the dynamic behavior of an organization [7].

As an example, the *data flow perspective* is formed when we impose relationships between three dimensions: activities, data items, and repositories. The data dependence mapping is one of the most interesting in this perspective. It reveals the data dependencies of an activity. For example, the review university admissions

application activity is data dependent on the repository that stores all of the university admissions applications, and the actual data within that repository.

Another example of interest is the *activity assignment perspective* formed by defining a set of relationships between three dimensions: employees, roles, and activities. Depending on the size and nature of an organization, the dimensions involved in the definition of this perspective can vary. For a small organization, with say 2 people and a relatively simple process, it is quite adequate and convenient to relate participants directly to the activities they perform. However, in organizations with large complex organizational structures and processes, this type of relationship is not practical; it is more appropriate to relate activities to roles, then relate those roles to participants. Therefore, through one level on indirection, activities are related to participants.

The probabilistic analyses that are a focus of this document can be applied to many different perspectives. Due to size limitations of this document, we will limit our further considerations to the process flow perspective. Process models of organizations are common and useful within complex enterprises. They are important for organizational understanding, analysis, and evaluation. They are also instrumental for design, deployment, and enhancement / evolution of workflow management systems. Streamlining, critical path analysis and workload analysis are all examples of useful analyses that can be performed using this perspective [2].

The classic definition of "basic ICN" within the process flow perspective [8] defines an AND/OR graph in which activity nodes, denoted by large circles, can have one incoming arc and one outgoing arc. All arcs are directed edges from one node to another, and represent precedence. There are two exceptions: the entry arc has no from-node, and the exit arc has no to-node. Parallelism and decision making are modeled by AND / OR fork and join control nodes. AND-logic nodes are denoted by small solid filled circles, and OR-logic nodes are small hollow circles. In this document, we extend the basic ICN definition by attaching probabilities to the arcs emanating from the fork nodes. See figure 1 for a simple Stochastic ICN example diagram. This definition of an admissible basic ICN includes restrictions that an ICN is single entry, single exit, and has properly nested AND / OR logic. We allow duplicate activity nodes and properly nested loops. A simple looping mechanism is implemented by a special type of OR node (loop termination) that has one of its two of its outgoing arcs pointing back to another special OR node (loop initiation) that has two incoming arcs and one outgoing arc. This is illustrated in figure 1 by the arc labeled 0.10 that allows arbitrarily many repetitions of the (compile references (C), evaluate references (E)) sequence to occur.

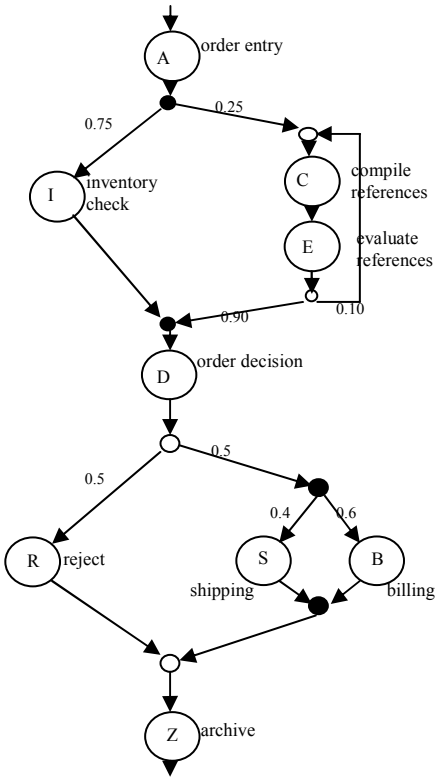

Figure 1: s-ICN of Order Processing

3. Stochastic Information Control Nets

There are a number of AND/OR graph models in the process modeling literature [12]. If AND/OR models are enhanced to be probabilistic, then even more types of analyses can be performed [13]. For example, a model may show decision making activity. Adding probability of selection to each possible decision choice creates an enhanced model that can be used to analyze congestion.

We create stochastic ICNs (abbreviated s-ICNs) by extending the classic ICN definitions. The basic ICN definition given above is extended as follows: The fork nodes of an ICN (both OR and AND nodes) are extended by attaching probabilities to the outgoing arcs; the probabilities attached to the outgoing arcs of any fork node must sum to one.

Figure 2 below shows the primitives that can be utilized to construct an s-ICN. Note that OR-fork and AND-fork have probabilities p_1 through p_n attached to outgoing arcs, but OR-join and AND-join do not have probabilities. A fork node can have any finite number of out-arcs (with $\Sigma p_i = 1$) which must be matched by the same number of in-arcs on the corresponding join node.

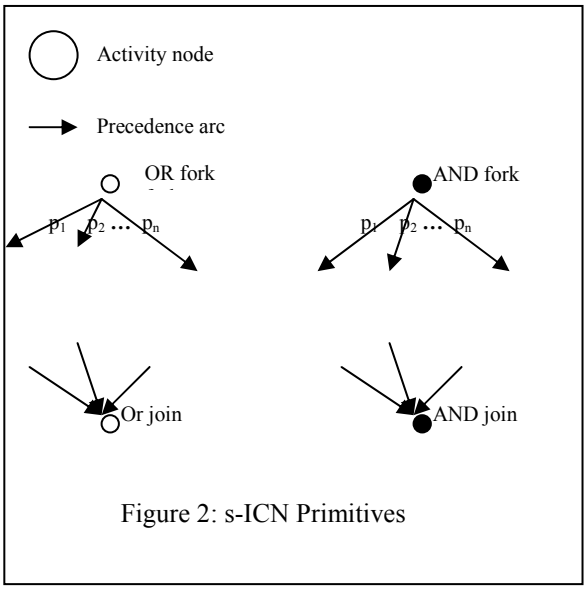

Figure 2: s-ICN Primitives

Figure 1 above shows an s-ICN of a simplified order processing procedure. It begins with order entry (activity circle labeled A) when a customer wants to purchase some goods. Notice that after this is parallel activity (AND split) because inventory check activity (labeled I) can proceed in parallel with the two reference activities (labeled C and E). Also note the back loop arc of C and E. This loop is possibly executed a large number of times, but the probability of this is quite small since the back arc is associated with probability 0.10. The AND join shows that the order evaluation activity, D, can only happen after both activity I and the C, E loop are finished. After D, either the customer is rejected (activity R) or shipping (S) followed by billing (B) happens. In any case, the activity labeled Z is the final activity.

Semantically, the OR split node represents an *exclusive or* function, so the probability attached to an OR arc represents the probability of selecting that arc as the execution path (and thus NOT selecting any other competing arc). Semantically, the probability attached to an AND arc represents the probability of selecting the activity at the tail of that arc for execution first (followed by execution of activities on other arcs and this arc in arbitrary order). As a simplifying abbreviation, arcs out of a fork node are allowed to be unlabeled, which means that the alternatives in that case are equiprobable. In all unspecified cases, alternatives are assumed to be equiprobable.

The attachment of probabilities to AND arcs is less straight forward than OR, has a novel semantic here, and has never been published in the workflow modeling literature. Consider the simple s-ICN shown in figure 3a which depicts the concurrent execution of two activities labeled S and B. The model shows that either shipping (activity S) or billing (activity B) can begin first, but both must occur. Since basic ICNs are defined such that no two activities are executed at exactly the same time (like Petri nets), there are only two possible strings (execution sequences), SB and BS. Note that this definition does not severely restrict modeling possibilities; if a long running activity (e.g. *shipping*), needs to be modeled, it can be done via two instantaneous activities labeled *start shipping* followed by *end shipping*. The probabilities on the two arcs emanating from the AND node each denote the likelihood that that arc is selected first. Since one or the other must be selected first, the probabilities on the outgoing edges of an AND fork node must also sum to one ($p_A+p_B = 1$).

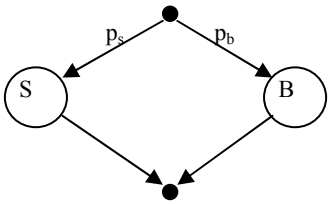

Figure 3a

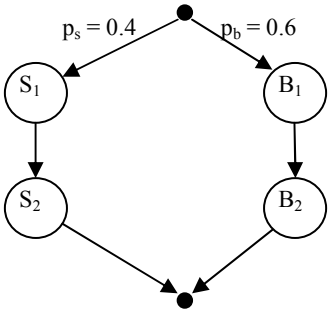

Figure 3b

If shipping and billing are complex operations, then we may want to split them into multiple activities. As another example, figure 3b shows an ICN of the AND combination of the two sequences S_1S_2 and B_1B_2. There are 6 possible execution sequences: three starting with S_1 and three starting with B_1 The total probability of the first 3 sequences must add to 0.40 and be equiprobable; thus the probability of any one of the three strings is 0.40X(1/3) = 0.133+. Similar logic holds for the other branch. Thus we can calculate 0.60X(1/3) = 0.199+ as the probability attached to the string $B_1S_1B_2S_2$. For the AND case, we are interested in counting the number of *admissible* permutations and their probabilities. For example we do not want to count the permutations such as $B_2S_1B_1S_2$ because it is not admissible for B_2 to precede B_1 according to figure 3b. We later present a recursive function to compute the number of admissible permutations in the general case. Note that this definition also does not severely restrict modeling possibilities; if we are concerned with attaching disparate probabilities to various strings, then we can always draw a more complex model to do this. In general, there is frequently a tradeoff between complexity and fidelity that should be carefully evaluated. [14].

4. Probabilistic Languages

Information Control Nets can be compared and contrasted with other models which are generators (e.g. phrase structure grammars, Petri nets) or acceptors (e.g. automata) of sets of strings or languages [20, 21]. An ICN is a generalization of a phrase structure grammar. A Stochastic Information Control Net generates (or specifies) a set of strings and each string has a probability associated with it. This construct can be viewed as a probabilistic language.

A Probabilistic Language (p-Language) has been defined in the literature as a set of strings, $\{s_i\}$, defined over an alphabet, *A*, in which each string, s_i, has a probability, $p(s_i)$, associated with it [21]. A p-Language is normalized iff summation $p(s_i)$ over all strings is one. In our model domain, the alphabet *A* consists of activity labels in an ICN process model, and strings represent possible execution sequences of a workflow depicted by the model. There are interesting results in the probabilistic languages literature that some normalized stochastic grammars generate probabilistic languages that are not normalized [20]; see figure 4 for an example. An important result established in

this document is that each and every probabilistic language generated by an admissible s-ICN is normalized. The s-ICN shown in figure 4 is not admissible because it is not well nested.

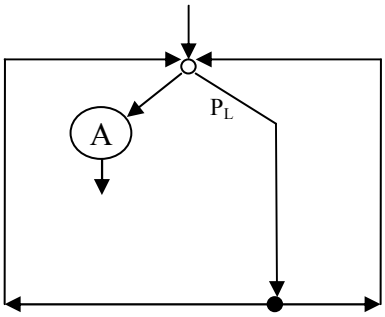

Figure 4: Non-normalized p-Language Generator

5. Model-Log Comparison

Workflow management systems generate workflow log files. Workflow models can be compared to workflow log files to discover anomalies, exceptions, activity evolution, security leaks, etc [5]. The workflow models sometimes represent the management view of processes. The log file may represent how the procedural work is actually done. It can be quite enlightening to discover how an organization really works, as compared to management's view of how the organization ought to work. All of the above investigations can be facilitated via *model-log comparison*. See, for example, the van der Aalst work on conformance checking [11].

Model-log comparison is executed via deterministic comparison of log entries to model execution sequences, or via probabilistic comparison of log entries to a probabilistic model. This document presents definitions, functions, and algorithms to allow both deterministic and probabilistic model-log comparisons via an s-ICN model.

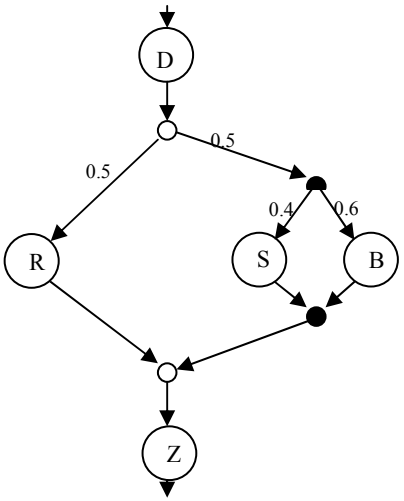

Figure 5

Figure 5 shows a simple s-ICN of concurrent shipping and billing surrounded by decision making to accept or reject the customer. Note that all execution sequences in this example begin with decision making (D) and end with archiving (Z). One execution sequence is customer rejection; this sequence consists of the 3 activities DRZ. If the customer is accepted, then the second arc of the OR node is traversed and there are two possible execution sequences: DSBZ (shipping is started before billing), or DBSZ (billing is started before shipping). Associated with any basic ICN model is a language defined as the set of all possible execution sequences. In the example of figure 5, there are three possible execution sequences, so the language generated by this model is {DRZ, DSBZ, DBSZ}.

Let's compare this model to a log. The log shown in table 1 below shows a history of activities for four customers: The first customer was accepted, and started billing before shipping (DBSZ), the second customer was rejected (DRZ), the third customer was accepted, and began shipping before billing (DSBZ), and the fourth was rejected (DRZ).

LOG OF CUSTOMER SEQUENCES

customer 1: DBSZ
customer 2: DRZ
customer 3: DSBZ
customer 4: DRZ

From this log and figure 5, we can do deterministic model-log comparison for model *correctness* and *completeness*. It is an easy matter to validate that all execution sequences of the model are in the log (correctness of the model), and that all customer entries in the log can be generated as execution sequences of the model (completeness of the model). This is a tiny toy example for exposition purposes. Note that it is not so easy to do this comparison by inspection if the model and the log are realistically large (a log typically has a very huge number of entries, some of which may be inaccurate).

Notice that, in figure 5, the OR-split (hollow small circle) has a probability of 0.5 attached to each of its two out-arcs. This means that about 50% of the customer cases go to the left (rejection), and 50% go to the right (shipping

and billing). We can utilize the probabilities attached to the OR-fork arcs (0.5 and 0.5) and the AND–fork arcs (0.4 and 0.6) of the example to do probabilistic model-log comparison to compute ***fidelity***.

By multiplying probabilities shown in figure 5 for each execution sequence in the model, we get the following probabilistic language for the model:

{(DRZ,0.5), (DSBZ,0.2), (DBSZ,0.3)}

It is also possible to derive a p-language from the log by labeling each entry of the log by its percentage of appearances. From the log in table 1, we can see that DRZ appeared in two out of four entries (50%), and ACDFG and ACEFG each appeared in 25% of the entries. Thus the log generates the following probabilistic language:

{(DRZ,0.5), (DSBZ,0.25), (DBSZ,0.25)}

Note that although both probabilistic languages have the same strings, they do not have the same probabilities, so the model does not have as much fidelity as might be hoped. The ***fidelity*** of a model with respect to a log can be mathematically computed by calculating the distance between the p-Language generated by the model and the p-Language derived from the log. There are numerous metrics that can be used to compute the distance between probabilistic languages including chi squared, earth mover distance, etc. The closer the distance is to zero, the better is the fidelity. If we change the probabilities on the AND-fork arcs in figure 5 to 0.5 and 0.5, then we obtain a complete fidelity model (zero distance).

To perform this comparison, we assume that name equivalencing of activities in the model and in the log has been completed. (e.g: all activities such as "billing" have the same label (B) in the log as in the model.) Also, creating the p-Language corresponding to a log is hard work, but it is a straight-forward normalization process as described above. Also, creating the p-Language corresponding to an s-ICN is hard work that can get quite complex if there is a lot of concatenation and nesting of split / joins nodes within other split / join nodes. In sections 7 and 8, we present the algorithm to generate a p-Language from an s-ICN, and show that the p-Language obtained is always normalized.

6. Related Work

Our work is concerned with ICNs, with probabilistic organizational models and languages, and with model-log comparison. There is a large literature covering these topics. We summarize and describe the relevance and similarity of some of that literature here.

+Probabilistic Languages

There is a rich literature concerned with stochastic grammars and the p-languages that they recognize [21] within computer science, mathematics, and linguistics. Our work has an intersection with work on Markov models and Bayesian networks. Also there is highly related work on probabilistic automata which are a generalization of Markov chains. The literature of probabilistic automata mathematically discusses many techniques and findings about, e.g. which p-Languages are generated by which types of automata [20]. This literature elucidates equivalences between stochastic grammars and probabilistic automata. It is interesting that there are non-obvious cases in the literature showing normalized stochastic grammars that generate p-Languages that are not normalized. Like ICNs, these representations are good for capturing sequence and OR behavior, but are different from the ICN model because they do not have any explicit notation to represent the parallelism of AND forks and joins.

+Probabilistic organizational analysis models

Numerous organizational analysis models are partially probabilistic in nature [6, 13]. These models are used in domains such as organizational efficiency analysis. Our work concerned with s-ICNs can be viewed as different, but complementary to these other works.

+Probabilistic workflow models

Within the area of org analysis, numerous workflow models, especially ones used in simulations [22], have a feature that allows the placement of probabilities, or probability distributions on the outgoing arcs of OR nodes. For example the IBM system, FlowMark, allowed the simulator to draw a model, and place probabilities on the

exclusive OR nodes. Also the ProM system at TU/e has plug-ins for many types of analyses. None of these systems have adopted our semantic to allow meaningful placement of probabilities on AND-fork nodes to extend to probabilistic languages.

+Petri net models

There are a number of AND/OR type graph models that have been presented in the workflow literature [12]. One popular workflow model is the Petri net [23]. Petri nets were defined as an abstraction such that execution of transitions is instantaneous, and token time on places is finite, but unspecified. There is an interesting literature concerned with stochastic Petri nets [22, 24]. These nets allow probability distributions specifying amount of time to be associated with transitions and/or places. This is different from the s-ICN which associates probabilities with the branches of OR-forks and AND-forks; our model is honed for organizational analysis and comparisons, and always generates normalized p-languages. The work of Varacca and Nielsen on Probabilistic Petri nets [16] is notably different from most of the stochastic Petri net literature; it is concerned with keeping track of probabilistic behavior, but does not employ timed nets. V+N define the notion of probabilistic word as a probability distribution over all strings of the same length, and then define a probabilistic language as a set of probabilistic words. Their model is concocted for different application purposes, and does not attack the important issue of normalization of probabilistic languages – this is more complex within their model.

+Model-log comparison

In the area of process mining, there have been publications that are concerned with the comparison of workflow models to event logs [4], and comparison of models to other models [3]. There have also been definitions of complexity, conformance, fitness, consistency, and completeness of models [14]. The work on conformance specifically compares a model to a log, and asks how well they conform [5]. The concepts of overfitting and underfitting are directly related to our concepts of completeness and correctness. Our model and algorithm are useful for the calculation of various probabilistic conformance metrics and to calculate fidelity. We note that none of these previous definitions have used probabilities on branches of OR-forks and AND-forks in the way that this document proposes to help quantify their analyses.

7. Normalized P-Language Generation

In this section, we present our algorithm to generate a p-Language from an s-ICN in the case of non-looping s-ICNs. In this algorithm, when we encounter fork and join nodes, we must combine strings, and attach probabilities to them. In this section we first present a recursive function to compute combined probabilities. Then we use this recursive function to do combinations. In this section, we use (s_i . $s_{j)}$ and (f_i X f_j). The dot notation denotes concatenation of strings, and the X simply denotes multiplication.

We need a general formula for calculation of the number of admissible permutations possible when joining a set of strings. This enables us to calculate the probability of each string. In general, an AND split can have an arbitrarily large, but finite, number of arcs emanating from it; if we assume no nesting of AND / OR nodes, then each arc represents a string. (Nesting will be incorporated later in the algorithm below.) We will denote the number of arcs as m. Also, a string s_i can have an arbitrarily large, but finite, number of symbols (activities) in it; we will denote the number of symbols in string s_i as its length, ln_i. Let the number of admissible permutations of m strings of length $ln_1, ln_2,\ldots, ln_m$ be $\Gamma(ln_1, ln_2, \ldots, ln_m)$. We can calculate the number of strings recursively using the observation that all strings must start with one of the m head symbols, and the number of strings starting with the k^{th} head symbol is $\Gamma(ln_1, ln_2,\ldots, ln_k\text{-}1,\ldots, ln_m)$. Summing these we get;

$\Gamma(ln_1, ln_2, \ldots, ln_m) = \Sigma\ (\Gamma(ln_1, ln_2,\ldots, ln_k\text{-}1,\ldots, ln_m)$ where the sum is from 1 to m.

Initial conditions for this recursion include:

$\Gamma(ln_1, 0) = 1$; $\Gamma(ln_1, l) = ln_1+1$

$\Gamma(ln_1, ln_2, \ldots, ln_{m-1},0) = \Gamma(ln_1, ln_2, \ldots, ln_{m-1})$

$\Gamma(ln_1, ln_2, \ldots, ln_{m-1},1) = (\Sigma ln_i+1)$ X $\Gamma(ln_1, ln_2, \ldots, ln_{m-1})$ where the sum is from 1 to m-1

Now we are ready to describe an algorithm to generate a normalized p-Language from an s-ICN assuming there are no loops in the s-ICN. Loops are covered in the next section. As an added bonus, our algorithm generates strings in probabilistic order! This becomes extremely important in nested looping s-ICNs because the generated p-Languages are infinite.

The algorithm traverses nodes of the s-ICN graph and manipulates a stack of contexts. Each context contains an ordered set of string fragments, and their probabilities. When a split node is encountered, the current context is suspended, and a set of new empty, active sub-contexts (one for each arc emanating from the split node) are created and pushed onto the top of the context stack. Each of these active contexts is expanded by traversing its sub-graph starting from its arc emanating from the split node until the corresponding join node is encountered. After all of the active sub-contexts are fully expanded, their string fragments are appropriately combined, and appended to their super-context. This super-context is then unsuspended, and its expansion is continued. The algorithm begins at the single entry node with a context of type global containing a single string fragment, λ, of length zero of probability one, and a level descriptor of value zero. The probability associated with this context is one. The algorithm terminates when it reaches the exit node at which time the entire p-Language is exactly the set of strings within the global context.

At every transition stage of our algorithm, we traverse an arc to reach a node within an active context in the stack, and then process the node. If the node is an activity node of the ICN, then we process it by concatenating the node label to all string fragments in the current context.

Whenever an OR split node is encountered, then we process it by suspending the current context, and creating a set of new empty, active contexts of type OR (one for each arc emanating from the split node) on the top of the context stack. Each empty context contains a single string λ of length zero and probability one. Note that each emanating arc has a probability associated with it designating the probability of traversing this arc; this is the probability that we assign to this context. The level descriptor for all of the new contexts is assigned a value of N+1, where N is the level descriptor of the suspended current context. We then proceed by traversing the first of the outgoing OR arcs (whichever is on the top of the stack – an arbitrary choice here) to reach the next node within the top active context in the stack, and then process that node.

Whenever an AND split node is encountered, then we process it by suspending the current context, and creating a set of new empty, active contexts of type AND (one for each arc emanating from the split node) on the top of the context stack. Each empty context contains a single string λ of length zero and probability one. Note that each emanating arc has a probability associated with it designating the probability of traversing this arc first among the outgoing arcs; this is the probability that we assign to this context. The level descriptor for all of the new contexts is assigned a value of N+1, where N is the level descriptor of the suspended current context. We then proceed by traversing the first of the outgoing AND arcs (whichever is on the top of the stack – an arbitrary choice here) to reach the next node within the top active context in the stack, and then process that node.

Whenever an OR join node is encountered, then we have finished processing one context. We check to see if all the sibling nodes have been processed. This check is done by comparing the level descriptor of this stack entry with the stack entry below it. If the level descriptor values are the same, then we proceed to process the next entry in the stack because it is a sibling node.

If the descriptor values are unequal (the stack entry below is N-1 in this case), then all the sibling nodes have been processed, so we can complete the siblings, and combine. For each sibling, we multiply each of its string probabilities by its context probability. Then for each string s_j in a sibling context, and each string s_i in the parent context, we create a new string $s_i.s_j$ within the parent context whose probability is the product $p(s_i)Xp(s_j)$. We sort these strings in order of decreasing probability. After all new strings are formed in the parent context, we can delete all old s_i and s_j strings, pop the children contexts from the stack, and re-activate the parent context. We then proceed by traversing the exit arc of the join node to reach the next node in the active parent context.

Whenever an AND join node is encountered, then we check to see if all the sibling nodes have been processed. This check is done by comparing the level descriptor of this stack entry with the stack entry below it. If the descriptor values are the same, then we proceed to process the next entry in the stack because it is a sibling node.

If the descriptor values are unequal (the stack entry below is N-1 in this case), then all the sibling nodes have been processed, so we can complete the siblings, and combine.

In this case, for each s_{i1} in context 1, s_{i2} in context 2, … s_{im} in context m (where m is the number of arcs into the AND join node), we want to generate all admissible combined permutation strings. If the length of each string s_{ik} is ln_k, then there are $\Gamma(ln_1, ln_2,\ldots, ln_m)$ admissible combined permutations (see above for the definition of Γ), and the length of each admissible combined permutation string is $\Sigma\ (ln_1, ln_2,\ldots, ln_m)$. To assign probabilities to these strings, we can calculate how many of these strings commence along arc k – there are exactly $\Gamma(ln_1, ln_2,\ldots, ln_k\text{-}1,\ldots, ln_m)$. These are the new strings that will be placed in the sibling context represented by arc k. Since these strings are equiprobable, and the total must be normalized, each of these strings will be assigned a probability of:

$P_k/\Gamma(ln_1, ln_2,\ldots, ln_k\text{-}1,\ldots, ln_m)$ where p_k is the probability assigned to the k^{th} context.

Then for each string s_j in a sibling context, and each string s_i in the parent context, we create a new string $s_i.s_j$ within the parent context whose probability is the product $p(s_i)Xp(s_j)$. We sort these strings in order of decreasing probability. After all new strings are formed in the parent context, we can delete all old s_i and s_j strings, pop the children contexts from the stack, and re-activate the parent context. We then proceed by traversing the exit arc of the join node to reach the next node in the active parent context.

Note that since the ICN is well nested, each join node encountered is guaranteed to match its corresponding fork node. In both of the above join processing cases, when we complete the siblings and combine, the sum of the probabilities is equal to the context probability of the parent context. Since the probability of the global context was initially defined to be one, the sum of the string probabilities associated with the exit node must be one. Thus, the generated p-language is normalized. It is also possible to expand our algorithm to cover loops. This case requires manipulation of infinite languages. This expansion is detailed in the next section.

8. S-ICNs with Loops

It is also possible to expand our algorithm to cover loops. This case requires manipulation of infinite languages. This expansion is detailed in this section.

Since the s-ICNs are well nested, we can represent them using the usual primitives: sequence, OR and AND split/joins, as well as the primitive LOOP. Any s-ICN can be expressed as the nested application of these primitives to a set of activities. Without loss of generality, we will assume that all splits are binary, and thus a single probability can be associated to them – say the probability associated to the first branch, since the other probability will be the complement to 1 on that figure. Given these restrictions, an ICN can be defined as:

– an activity A
OR(ICN1,ICN2,p), where p is the probability that the OR will choose the first branch;
AND(ICN1,ICN2,p), where p is the probability that the first activity to be executed is the first one in the first branch;
SEQ(ICN1,ICN2);
LOOP(ICN1).

A p-language set is a set of words with their associated probabilities <w,pw>

The generation of the p-language of an s-ICN is recursively defined as

1) if the ICN is an activity A its plang(ICN) = {<A,1>}

2) if the ICN = OR(ICN1,ICN2,p) then plang(ICN)= plang(ICN1)*p U plang(ICN2)*(1-p)
where we define plang(ICN)*p is the set {<x,px*p> | <x,px> in plang(ICN)}

3) if ICN = SEQ(ICN1,ICN2) then plang(ICN) = {<xy,px*py> | <x,px> in plan(ICN1) and <y,py> in plang(ICN2)}

4) if ICN = AND(ICN1,ICN2,p) then plang(ICN) = { <I(x,y),p*px*py/T(|x|-1,|y|)>} U {<I(y,x) , (1-p)*px*py/T(|x|,|y|-1) } where <x,px> in plang(ICN1) and <y,py> in plang(ICN2) , the I(x,y) operation

generates a string that is any interleave of x and y starting with the first activity being the first activity of x, and T is the function defined above on the number of interleaves.

5) if ICN = LOOP(ICN1) then plang(ICN) = U plang(ICN1)^k for k ranging over all positive integers
= {plang(ICN) U (plang(ICN)* plang(ICN)) U (plang(ICN) * plang(ICN) * plang(ICN)) U …}

- Note that in each of the 5 cases above, since ICN1 and ICN2 are normalized, it follows that the resultant ICN is also normalized. For example, in the loop case, the probability of the resultant = (pr no loop) + (pr loop once) + (pr loop twice) + … = $(1-p^L) + ((1-p^L)*p^L) + ((1-p^L)* p^L* p^L)$ where p^L is the probability attached to the loop back arc. Summing this expression, all of the p^Ls cancel yielding a sum of 1; thus the resultant is normalized. Figure 4 illustrates a simple case of an inadmissible s-ICNs whose p-Language is not normalized.

9. Maximal String Generation

Since an s-ICN with loops generates an infinite p-Lang, we do not actually write down or print out every string of the language. In this section we show an algorithm to generate strings (as many as we want) in most-probable-string-first order. We first show how to generate the maximally probable string of any s-ICN. Then the essence of our algorithm is to eliminate this string, and re-apply the algorithm to generate the next-best string, repeatedly. This allows us to generate finite approximation languages with arbitrarily high fidelity.

1) if the ICN is a single activity A its maximal string is max(plang(ICN)) = {<A,1>}

2) if the ICN = OR(ICN1,ICN2,p) then max(plang(ICN)) = max(max(plang(ICN1)*p), max(plang(ICN2)*(1-p))) where we define plang(ICN)*p is the set {<x,px*p> | <x,px> in plang(ICN)}

3) if the ICN = SEQ(ICN1,ICN2) then max(plang(ICN)) = <xy,px*py> | <x,px> is max probability string in plang(ICN1) and <y,py> is max probability string in plang(ICN2)}

4) if ICN = AND(ICN1,ICN2,p) then max(plang(ICN)) = the maximal product string in the set { <I(x,y),p*px*py/T(|x|-1,|y|)>} U {<I(y,x) , (1-p)*px*py/T(|x|,|y|-1) } where <x,px> in plang(ICN1) and <y,py> in plang(ICN2) , the I(x,y) operation generates a string that is any interleave of x and y starting with the first activity being the first activity of x, and T is the function defined above on the number of interleaves.

5) if ICN = LOOP(ICN1) then max(plang(ICN)) = the maximal string in the set {U plang(ICN1)^k} for k ranging over all positive integers. This could be problematic since the set is infinite; however, we only need to consider the single iteration strings (plang(ICN1)), and this is finite. Note that any string which traversed the loop twice has a smaller probability by at least p^L than a prefix of the string which did not loop back. So every string that traversed the loop-back path is non-maximal.

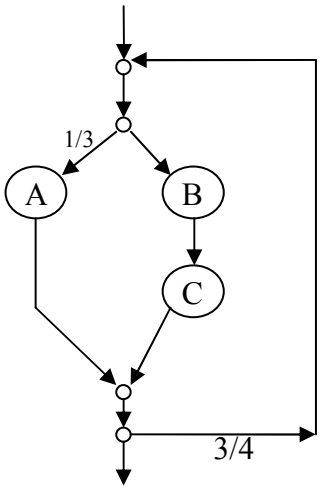

Figure 6: Stochastic ICN with Loop

A looping s-ICN is illustrated in figure 6. Its p-Language contains:

{A, 1/3 x 1/4;
BC, 2/3 x 1/4;
AA, 1/3 x 3/4 x 1/3 x 1/4;
BCBC, 2/3 x 3/4 x 2/3 x 1/4;
ABC, 1/3 x 3/4 x 2/3 x 1/4;
BCA, 2/3 x 3/4 x 1/3 x 1/4;
AAA, 1/3 x 3/4 x 1/3 x 3/4 x 1/3 x 1/4;
…}

Although there are an infinite number of strings in this language, we immediately know that the maximally probable string is in the set of non-repeating strings, so only strings A and BC are candidates. We compare their probabilities, and choose BC as the max. Note that although the string A is shortest, it is not maximal. Also, the string BC repeated twice is actually equally probable as the string A.

This algorithm for generating the maximal string also leads to an algorithm for generating the maximal K strings by eliminating this maximal string, and re-applying the algorithm to generate the next-best string, repeatedly. It is not difficult to handle any combination using the rules above including loops nested within loops. There are details of this algorithm which we do not cover here. For example, when a string is removed from the set of single iteration strings, others must be added – e.g: $(\text{max string})^2$ as in figure 6; this is tedious, but a finite process.

10. Summary and Conclusions

In this document, we extended the notions of classic Information Control Nets [7, 8] to define new concepts of Stochastic Information Control Nets. We introduced a novel and useful AND-probability semantic and show how this probabilistic mathematical model can be used to generate normalized probabilistic languages. We presented a new definition of fidelity of a model. These definitions can be applied to many aspects of organizational modeling including the process, the informational, and the interpersonal perspectives. The document also proved that the algorithm generates probabilistic languages that are normalized, and in probability ranked order.

The algorithms presented are finite but at times "messy." Ongoing work is concerned with decreasing this messiness, and calculating the computational complexity of the algorithms. There are interesting special cases where

the computation can be less messy and short-cutted - for example if all branches are equiprobable. Other useful theorems and algorithms:

1. Given any string within a s-ICN's language, it is computationally simple to compute its probability
2. Given any s-ICN with loops, that generates a p-Language L, we can always derive an approximation language L* such that |L-L*|<e

11. Acknowledgements

The authors would like to thank our colleagues, and our institutions for sponsoring this research.

12. References

[1] Aalst, W.P.M. and Song, M., "Mining Social Networks: Uncovering interaction patterns in business processes", in Business Process Management, pp. 244-260, 2004.

[2] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede (eds), Process Aware Information Systems, John Wiley and Sons, New Jersey, 2005.

[3] W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters, "Process Equivalence: Comparing Two Process Models Based on Observed Behavior", Lecture Notes in Computer Science, Springer, 2006.

[4] J. E. Cook and A. L. Wolf, "Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model", ACM Transactions on Software Engineering and Methodology, 8(2) : 147–176, 1999

[5] W.M.P. van der Aalst, "Business Alignment: Using Process Mining as a Tool for Delta Analysis and Conformance Testing", Requirements Engineering Journal, volume 10, issue 3, pages 198-211, 2005.

[6] Drasgow, F. and Schmitt, N., Measuring and Analyzing Behavior in Organizations, Prentice Hall, 2001.

[7] Ellis, C.A. "Formal and Informal Models of Office Activity" in Proceedings of the IFIP International Computer Congress, Paris, 1983.

[8] Ellis, C.A. "Information Control Nets: a Mathematical Model of Office Information Flow" in Proceedings of the Ninth Annual ACM Conference on Simulation, Measurement, and Modeling of Computer Systems, August 1979.

[10] Rembert, A.J. "Automatic Discovery of Workflow Models" PhD dissertation, University of Colorado at Boulder, Computer Science Department, 2008.

[11] Rozinat, A. and W. M. P. van der Aalst, "Conformance checking of processes based on monitoring real behavior," Information Systems, vol. 33, no. 1, pp. 64–95, March 2008.

[12] Sinnakkrishnan Perumal† and Ambuj Mahanti, "Applying Graph Search Techniques for Workflow Verification" Proceedings of the 40th Hawaii International Conference on System Sciences, January 2007.

[13] Laguna, M. and Marklund, J., Business Process Modeling, Simulation, and Design, Pearson Prentice Hall, New Jersey, 2005.

[14] I. Vanderfeesten, J. Cardoso, J. Mendling, H. Reijers, W.P.M. van der Aalst, "Quality Metrics for Business Process Models" in BPM and Workflow Handbook, L. Fischer (ed), 2007.

[15] J. Desel, G. Juhas, R. Lorenz, and C. Neumair, "Modelling and Validation with VipTool" in BPM and Workflow Handbook, W. van der Aalst, A. ter Hofstede, and M. Weske (eds), 2003.

[16] Varacca, D. and Nielsen, M., "Probabilistic Petri Nets and Mazurkiewicz Equivalence", Citeseer website: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.303. 2003.

[17] G. Balboa "Introduction to Stochastic Petri Nets" in Euro Summer School on Trends in Computer Science, v.2090 of LNCS, pp. 84-155, 2000.

[18] I. Vanderfeesten, J. Cardoso, J. Mendling, H. Reijers, W.P.M. van der Aalst, “Quality Metrics for Business Process Models” in BPM and Workflow Handbook, L. Fischer (ed), 2007.

[19] A. Rozinat, R.S. Mans, M. Song and W.M.P. van der Aalst, “Discovering Simulation Models” Information Systems, Vol. 34, pp 305-327, 2009

[20] Rabin, M.O.,"Probabilistic Automata", Information and Control **6**, pp.230-245, 1963.

[21] K. S. Fu, “Stochastic Languages, Stochastic Automata, and Pattern Recognition”, Defense Technical Information Center Report #AD0719801, 1990.

[22] L. Fischer (ed.) The 2007 BPM and Workflow Handbook, Future Strategies, Inc., 2007.

[23] W.M.P. van der Aalst and K. van Kee, Workflow Management – Models, Methods, and Systems, MIT Press, 2000.

[24] Bause, F. and Kritzinger, P. S., Stochastic Petri Nets, 2nd ed., Vieweg, 2002.