

Simulating Design Processes with self-iteration activities based on DSM planning

Arie Karniel
ariek@eng.tau.ac.il

School of Mechanical Engineering, Tel Aviv University, Tel Aviv, Israel

Yoram Reich
yoram@eng.tau.ac.il

Abstract

Design Processes are fundamentally iterative. The increasing knowledge about the product, while designing, manifests in design changes of previously accomplished activities. Such iterations are considered as a major source of increased product development lead-time and cost; thus process planning and simulation are essential.

The Design Structure Matrix (DSM) is a known method for process planning. Since the DSM itself does not express all the relevant information that is required for defining process logic, process logic interpretation is required. "Business Rules", being logic interpretation options that are applicable in different business cases, can guide automatic translation of the DSM to valid iterative process plans.

This study focuses on implementing self activity iterations in process simulation. Notwithstanding their importance, they are overlooked in DSM literature. Using different business rules results in various Run time processes that exhibit non-trivial behaviors. Interpretation rules are fully validated for a test case.

1. Introduction

Product Development Processes (PDP) can be defined as collection of related activities targeted to converting a new concept (market opportunity) into a marketed product. Within PDP, the cyclic Design Process (DP) incorporates synthesis, analysis, and decision-making stages to migrate from problem definition to artifact specification that meet the marketing and business case vision. The Design Process reflects the specific design requirements, objectives and constraints applicable to the product.

Iterations are inherent to DPs, as changes in the design of one component may result in changing the design of other components. Additionally, the process includes testing activities that are a-priori planned iterations [14] and unpredicted feedback rework due to unplanned changes or problems. Iterations are considered as major source of increased product development lead-time and cost [6][9].

One type of iteration occurs when an activity is iterated several times before completion. Self-iteration of a design activity could always be separated to

design, check and decide. Such distinction, (and subsequently, proliferation in the Design Structure Matrix (DSM)), might be justified if the distinct activities are performed by different resources, but might be just an overhead if done by the same designer or design resource. If all iterations are completed before continuing to the next activity, the only impact of self iteration might be duration extension. However, if such self-iteration is executed in parallel to other activities, its influence span may extend to all linked activities that are executing or completed execution between the completion of last iteration and the current one. Self activity iterations are briefly mentioned in [10], but the implications of including activity self iterations within concurrent process implementation, are somewhat overlooked in the DSM-based simulation literature. The inclusion of self iterations, elicits a wide range of potential Run Time process structures. Some of the additional process characteristics are manifested in the cases of overlapping parallel activities [7].

Planning the process and performing project activities in an appropriate sequence is critical in order to minimize rework due to information dependencies. Introduced by Steward [8], the DSM is utilized to capture the required product knowledge for process planning. Yet, the DSM does not represent any information regarding the linkages logic, and the presented information is insufficient for implementing process simulations [8][23].

Process verification issues being thoroughly discussed in process modeling literature [2][18], are seldom mentioned in the DSM literature [12]. Verification checks are essential for automatic translation of DSM to concurrent process plan model, the Process Scheme. Translating the DSM is not unique due to interpretation options based on the applicable business cases [13]; these interpretations defined as Business Rules may impact the process scheme or merely the actual Run Time processes.

In what follows, section 2 surveys the DSM and process verification literature and section 3 merges them into the verification of DSM-based process plans. Section 4 analyzes implementing the method from section 3 to the case of self iterations for a simple case study. This analysis elucidates the behavioral richness and the complexity of implementing self-iterations. Conclusions are drawn in section 5.

2. Literature survey

The term *Activity* is used to describe a logic step within a process. Activities have pre and post conditions; additionally, they may accept inputs and share outcomes during their performance [7]. The term *Process Scheme*, is used to describe the process structure, i.e., the model of linkages between activities, their precedence, concurrency, and logic relations [12]. The term *Run Time process*, is used to describe the process progress according to the *Process Scheme*.

A limited set of process constructs are being used to define the Input logic of an activity (pre-conditions) and its Output logic (post conditions). Karniel and Reich [12] used the construct definitions (Workflow patterns) elaborated in [3]. The constructs utilized in this study are¹: Serial link {1}; Split-And {2}, sending a termination signal through all output links; Split-Or {6}, sending termination signal to some of the output links; Split-Xor {4}, sending to only one of the output links; Join-And {3}, wait for all input signals; Join-Or {8}, wait for first to come, execute multiple times; and Join-Xor {5}.

2.2. Process planning using DSM

The commonly used GANTT and Pert charts are inadequate for planning design processes, as they do not effectively model design activities dependencies and process iterations [23]. The DSM representation can be used for various applications [5]. The DSM provides the means to identify serial, parallel and iterative design flows; model and manipulate iterations and multidirectional information flows [9]. DSM is square matrix that uses off-diagonal entries to signify the dependency of one element on another. When modeling processes, the lower diagonal portion represents a precedent activity relationship; i.e., a marking in cell $\{j,k\}$ (row j , column k , $j > k$), indicates that activity k should follow activity j . For each activity, its row shows its inputs and its column shows its outputs. The upper diagonal portion of the DSM matrix denotes an iterative process (a link to an upstream activity). The outcome of a particular activity can directly impact a previously performed activity, which should be performed again, i.e., rework.

Serial activities are linked by forward link, and indicate a serial process, Figure 1(a). Parallel or concurrent activities have no relation linkages, Figure 1(b). Coupled activities have forward and feedback links, Figure 1(c). Additional relation type, *Overlap*

activities can be considered as a Parallel relation [24]. Cho and Eppinger [7] model two overlap activities as parallel activities with additional lead time, and a constraint preventing completion of the latter activity before the first activity. Coupled activities form an activity loop (cycle). When the cycle includes three activities (and more), the cycle may have many forms in the DSM representation [12].

Different DSM markings are typically used. *Binary DSM* uses tick-marks ('x') to represent precedence relations [22], (e.g., Figure 1). *Numeric DSM* represents a measure of the degree of relation or its importance ranking. *Probability DSM* [21] is used in this study. Probabilities are assigned directly or result from transformation of Numeric (or Binary) DSM [19][23]. The reordering algorithms: Partitioning, Sequencing, and Clustering typically do not utilize diagonal values. The diagonal values are typically left empty, or used for describing activity properties (e.g. Duration). In this study the diagonal values are used for self-iteration probabilities.

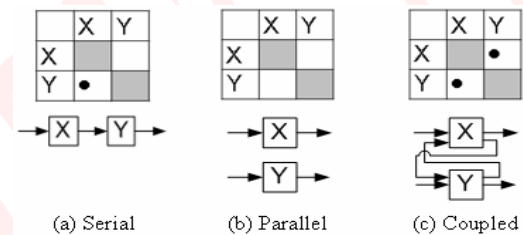


Figure 1. Activity relations

The information presented in the DSM is insufficient for simulation, e.g., activity duration. Such information should be specified in addition to the DSM. Since such information is not included, algorithms that use only the DSM structure, are criticized as inadequate for optimization, and require simulation-based optimization[6].

Karniel and Reich [12] surveyed various simulation process structures and process progress types, including Deterministic progress, Markov chain or Monte Carlo simulations. The survey analyzed the translation of DSM-based process plan, into process simulation in various studies, classified the approaches used, and discusses their strengths and limitations along problems related to process modeling verification.

2.3. Process verification

Process correctness verification is profoundly discussed in the Workflow literature [16][18]. An established formal languages model for process specification is Petri net [15], which is a bipartite

¹ The numbers in bracket indicate the definitions in [3]. Synchronizing Merge {7} is not used in the current study.

directed graph with two node types called places and transitions, connected by directed Edges (arcs). Places can contain Tokens. A transition may transfer tokens from its input places to its output places according to rules. Process activities are modeled by transitions; places correspond to conditions. Marking, representing the process state, is the distribution of tokens over the places. Petri nets can be used as diagrammatic tool for modeling and analyzing distributed system behavior including sequential, conditional, parallel and iterative routing [2]. WF nets (workflow nets) [1] are Petri nets, used for specifying the control of workflow processes with defined starting and ending. Using the formal definitions of a WF net one can set the required conditions for proper process specification (*Process Scheme*), using a correctness criteria named 'Soundness'. WF net properties are [1]: 1) A WF net has starting place and final place. 2) All the activities and places are on a path from the starting place to the final place (i.e., no activities or conditions that do not contribute to process progress).

Proper process, defined according to the soundness criteria has the following properties. 1) From every process state, which is reachable from the initial state, (token at the starting place); there is a 'firing sequence' that leads to termination state. The process should terminate eventually. 2) Once the terminal state was reached there are no activities that did not perform yet; formally: there are no tokens in places other than the final place (i.e. the termination state). 3) There are no inactive activities which could not execute due to unmet pre-conditions

3. Method description

This section briefly describes the method and definitions in [13] that are utilized for the analysis and examples in section 4. Based on the influence of changing product parameters in a design activity on other design activities, an impact DSM is defined. Then it is converted to Probability DSM; and finally reordered [11]. Unlike the typical DSM, diagonal elements that indicate self-iteration probability, (i.e., the probability that activity should be reworked again) are estimated and added. Self-iteration imply that the work done was not approved, e.g., some aspect of the design was not approved by the reviewer, or a piece of software code did not pass its unit test (e.g., in Extreme Programming (XP) [4]).

Translating the ordered DSM into a Process was done (Figure 2): (1) Translating the DSM into a Design Process Matrix (DPM); adding logic activities. (2) Converting DPM into an equivalent Current process

scheme (C- Process). (3) Using the process scheme for creating a Run Time process (RT- process).

3.1. Validity requirements

The basic validity requirements presented were:

(1) The Product Design Process has the characteristics of a project, thus the process should have a defined start and defined end.

(2) From any given process status, the process should be able to reach the process end (i.e., reaching End activity which implies process end status).

(3) When the process reached the termination state (End activity executes), this should imply:

- a. all the Design Activities (and all their iterations) have completed; and
- b. each activity has been performed at least once.

(4) The process should be traceable.

(5) Despite the iterative nature of the process, which enables infinite loops, the process should be enforced to complete in a finite time.

3.2. Implementation Rules:

The implementation rules may have sub options that are defined as Business Rules (BR), i.e., they should be chosen according to the business environment and business constraints considerations (e.g., Time versus Resources). A design activity X has duration property and it may iterate. The activity duration, $duration(X)$, is not described by the DSM matrix. The self-iteration probability is $p=P_x$, on the DSM diagonal. Translating a single activity DSM to a DPM implies adding the logic activities: Begin (B), End (E), Input logic (IL), and Output logic (OL).

The following *Implementation Rules* (IR) apply:

(IR 1): Logic activity duration is zero; it does not have self-iterations.

(IR 2): Design activity has one forward input links (from IL) and one forward output link (to OL).

(IR 3): Logic activities may have multiple input or multiple output links, but at least one forward input link and one forward output link.

The case of one activity is depicted in Figure 2. The initial one activity DSM (a) is translated to DPM (b). Logic activities are added: B, E, IL, and OL. The marking 1 in the DPM represents a link with probability $p=1$. The self-iteration feedback probability $p=P$, indicates the iteration of X , and is set between the Output logic activity and the Input logic activity.

The logic applied in the IL is Join-Or, i.e., signal from B or a feedback signal from the OL. Respectively, the logic applied to the OL (decision procedure) is Split-Xor; either sending feedback or a signal to the E.

The resulting Current process scheme, Figure 2(c), is directly inferred from the DPM. Each off-diagonal element becomes a link. Input logic (i) and Output logic (o) are explicitly indicated, (they will be implicitly assumed in following figures). Due to the possible iterations, the Run Time process in Figure 2(d) has potentially infinite number of configurations, according to the number of repetitions. In [13], the number of iterations is practically limited by setting a threshold on the feedback probability. Setting P_{min} as threshold derives the maximal number of iterations, where $ceil$ function is the nearest greater integer.

$$N_{max} = ceil(\log(P_{min})/\log(P)) .$$

Two activities may be parallel independent, serial or coupled. The parallel independent case induced the following IRs, being introduced as a procedure that handles multiple parallel initiation and parallel termination of multiple paths simulation [13].

- (IR 4): The logic of Begin activity is Split- And.
- (IR 5): The logic of End activity is Join-And.
- (IR 6): If an activity has no previous source (in link), it should be linked from the Begin activity.
- (IR 7): If an activity has no target (out link), it should be linked to the End activity.

The following symbols were defined [12]: Logic indication: Split (\Rightarrow) for Output logic; Join (\Leftarrow) for Input logic. Logic operations: + (OR); \bullet (AND); \oplus (XOR, Exclusive-Or).

The short form of multi-variable logic operations: Multiple-Or $\Sigma(A_i) = A_1 + A_2 + \dots + A_n$; Multiple-And $\Pi(A_i) = A_1 \bullet A_2 \bullet \dots \bullet A_n$; and Multiple-Xor $\otimes(A_i) = A_1 \oplus A_2 \oplus \dots \oplus A_n$, where A_i represents a link signal, which can be a forward link F_i or Iteration (feedback) link I_i . Using the above symbols, the following formulation was presented for the implementation of Begin and End:

$$Begin \Rightarrow \Pi(F_i) \quad (1)$$

$$End \Leftarrow \Pi(F_i) \quad (2)$$

Forward links and feedback (iteration) links may have distinct logic operands; the following basic rules were defined for both IL and OL [13].

- (IR 8): Forward links to design activities (lower part of the matrix) have AND logic, on first iteration.
- (IR 9): Forward link to the End activity, have XOR (Exclusive OR) logic with the other links.
- (IR 10): Feedback (iteration) links have OR logic.

In simple cases, the Input logic defines accepting signal from all forward links (Join-And), i.e., the activity starts once all its precedent activities have completed; or a signal from any of the feedback links is available; or both, (cf. Eq. 3). The Out logic procedure may send signals to any feedback link, or (exclusively)

send signals to all forward links, but not both (cf. Eq. 4).

$$IL \Leftarrow (\Pi(F_i)) + (\Sigma(I_i)) \quad (3)$$

$$OL \Rightarrow (\Sigma(I_i)) \oplus (\Pi(F_i)) \quad (4)$$

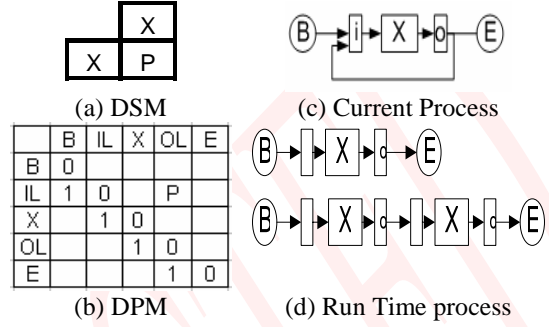


Figure 2. Full process description

3.2.1. Serial activities. The validity requirement 3b (in Section 3), indicating one execution at least, imposes different logic requirements on the first execution of the design activity versus further iterations. For example on its first execution, the activity cannot send forward signal to End activity; sending a forward signal to the next serial activity is required, otherwise the next activity will not be performed. Sending signals to both hinders IR 9, and was shown to cause invalid processes.

Other implementation issues have several options that are defined by Business Rules (BRs) numbered as sub cases of the IRs (see appendix A for relation of BRs and logic formulation equations).

- (IR 11): OL options: sending completion signal to following serial design activities by:
 - (BR 11.1): Sending only once the activity has completed all its iterations; or
 - (BR 11.2): Sending once the activity has completed its first iteration (i.e., early start of next activity). Yet, sending a completion signal to E can be done only once all its iterations have completed (i.e., IR 9).
- (IR 12): OL: signal to E. Second (or later) iteration of an activity (e.g., X), may or may not send signal to E, while the following serial activities (e.g., Y) have started execution (or have completed):
 - (BR 12.1): On second (or later) iteration, the activity must be followed by its next serial activities.
 - (BR 12.2): On second (or later) iteration of the activity, the next activities may follow, or the E may follow (not both, subject to IR 9).

There are 4 BR combinations that change the OL, and are manifested in the DPM. The first was the case of BR 11.1 and BR 12.1, which was already described

by Eq. (4). The combination BR 11.1 and BR 12.2 is formulated in Eq. (5); BR 11.2 and BR 12.1 in Eq. (6); and BR 11.2 with BR 12.1 in Eq. (7).

$$OL \Rightarrow (\Sigma(Ii) \oplus \Sigma(Fi)) \oplus \text{End} \quad (5)$$

$$OL \Rightarrow (\Sigma(Ii) + \Pi(Fi)) \oplus \text{End} \quad (6)$$

$$OL \Rightarrow (\Sigma(Ii) + \Sigma(Fi)) \oplus \text{End} \quad (7)$$

Allowing early start option (BR 11.2) derives Run Time options (IR 13) not described by the DPM.

(IR 13): Iterations of same activity cannot co-occur:

(BR 13.1): While the activity is executing, link signals are directed to this activity (i.e., to its prior IL).

(BR 13.2): While the activity is executing, link signals are directed to next iteration of the activity (which cannot start until current iteration ends).

In the context of administrative processes, such option was defined as Lack of Synchronization conflict [17]. The first option (BR 13.1) entails defining the consequences of the additional iteration rework. Typically, the activity duration will increase. Recalculating the activity duration in this case is similar to the overlapping activities case [7], and can be addressed by the same approaches.

3.2.2. Coupled activities. Coupled activities may have serialization requirements (e.g., testing should follow design activities). Starting coupled activities in parallel result in processes whose OL is similar to the case of BR 11.2 (parallelization of the iterations of serial activities); but with different IL.

(IR 14): Coupled activity execution start:

(BR 14.1) (serialization): coupled activity may start, after its previous activity (according to DSM) has completed at least once.

(BR 14.2) (parallel): coupled activity may start in parallel to all other activities in the same loop.

The Input logic for BR 14.2 is formulated in Eq. (8), (replacing Eq. (3)); and is equivalent to the parallel process case [20].

$$IL \Leftarrow \text{Begin} + \Sigma(Ii) + \Sigma(Fi) \text{ in loop} \quad (8) \\ + \Pi(\text{other forward links})$$

The options described in IR 12, are respectively replaced by the options of IR 15.

(IR 15): Sending signal on completion of a coupled design activity is done according to one of the following business rules:

(BR 15.1): A coupled design activity should link to the next activity on its completion.

(BR 15.2): A coupled design activity may link to the End activity on its completion.

4. Examples and results

A comprehensive interpretation of the DSM-based plan is given for the case of two design activities. First,

an enumeration approach is used to generate potential processes, and filter those that comply with the validation rules. Then, examples of process logic problem are depicted, justifying some of the rules.

4.1. Validating the case of two design activities

Applying at most one iteration (maximum 2 executions), results in 74 distinct valid Run time processes, which exhibit some non-trivial behavioral aspects. The validation analysis is done using the implementation rules applied to all potential 128 process combinations.

4.1.1. Run Time cases enumeration. For maximal number of iteration $N_{max}=0$, we have exactly one execution of each design activity (as enforces by validation rule 3(b)). The number of activities in this case is exactly four (Begin, X, Y, End). Process generation options for building a Run Time process with two design activities are summarized in Figure 3.

Link Option	Process scheme	Description	Applicable rules/ Eq.
BX	B-X	Default link to first activity from B	IR 6 + BR 14.1 / Eq. (3)
Bs	$\begin{matrix} B-X \\ \swarrow \\ Y \end{matrix}$	Split: from Begin to multiple activities in parallel	IR 6 + BR 14.2 / Eq. (8)
XY	X-Y	Final iteration of an activity links to next activity	BR 12.1 or BR 15.1 / Eqs. (4), (6)
XE	X-E	Final iteration of an activity can link directly to E	BR 12.2 or BR 15.2 / Eqs. (5), (7)
YX	Y-X	Final iteration of an activity which can feedback to other activity in loop	BR 14.2 / Eqs. (6), (7), (8)
YE	Y-E	Final iteration of an activity without target should link to E	IR 7 / Eqs. (4), (5), (6), (7)

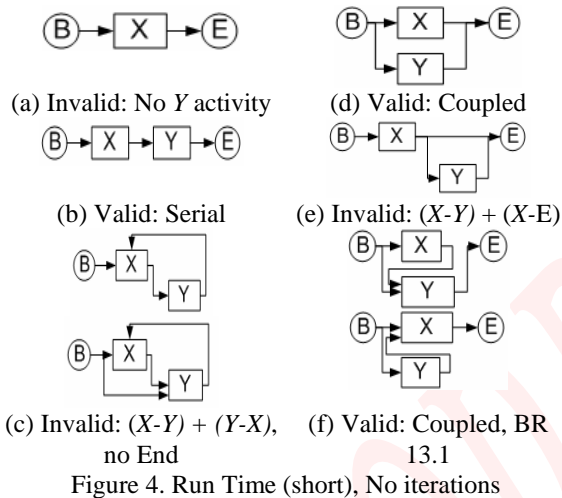
Figure 3. Optional run time linkages

There are two options for starting the process, link from B to X (BX), or split to both design activities (Bs); two options to link X after its completion: link to E (XE), or link to Y (XY); and similar options for Y (YE, YX). The options presented for X and Y are the only applicable options of the final iteration of an activity (i.e., it can not iterate to itself in this case, $N_{max}=0$). Some options are applicable only for specific business rules and applicable equations, as indicated.

There are eight potential Run Time processes, utilizing the three choices: Begin (2 options), X (2

options), Y (2 options). However, validity requirements filter out some of them. The examples are depicted in Figure 4. For easier display, we assume the Input and Output logic are embedded in the activity (short RT process description).

The option $(BX+XE)$ is invalid since Y activity is not executed. Using '*' as don't care symbol, we can define the choices as $\{BX, XE, '*'\}$, i.e., the process is equivalent for both choices of Y option. In general, any serial process combination, which includes only iterations of activity X is invalid, Figure 4(a).



The serial valid case is depicted in Figure 4(b). The combination of options $(XY+YX)$, Figure 4(c), is always invalid since the process does not reach the End activity, violating validity requirement (1). This combination occurs for two cases of process start $\{ '*', XY, YX \}$.

Valid Parallel options are depicted in Figure 4(d) and (f). In the latter case there should be a difference in the activities duration, applying rule BR 13.1 (rule BR 13.2 cannot apply since there are no additional iterations). If both activities have exactly the same duration, the options in Figure 4(f) are reduced to the case in Figure 4(d). The Run Time process cases described above are all the eight potential cases derived from the options in Figure 3.

An additional invalid option, which might graphically look like other Run Time processes, is depicted in Figure 4(e). Such option cannot be generated from the options presented in Figure 3; X cannot link to both Y and End. Using business rules interpretation, proceeding to End activity cannot be done in parallel to proceeding or feedback to other activities (Xor logic). Proceeding to End should mean completion, while feedback or continuing to other activities has a contradicting intent.

4.1.2. Run Time with iterations. Allowing one iteration of each activity, $N_{max}=1$, either self-iteration or iteration due to the other activity, yields maximum six activities: Begin, first X iteration, first Y iteration, last iterations of X and Y , End. In addition to the linkage options described in Figure 3, the interim iterations (i.e. first activity execution) have total of four options detailed in Figure 5.

Link Option	Process scheme	Description	Applicable rules/equations
XX	X-X	Self-iteration	Eqs. (4), (5), (6), (7)
YY	Y-Y	Self-iteration	Eqs. (4), (5), (6), (7)
Xs	$\begin{matrix} X-X \\ Y \end{matrix}$	Split: Iteration and forward link, using the applicable business rules	BR 11.2 or BR 14.2 Eqs. (4), (6)
Ys	$\begin{matrix} Y-X \\ Y \end{matrix}$	Split: Choosing multiple iterations (feedback links have OR logic)	Eqs. (4), (5), (6), (7)

Figure 5. Additional linkage options for iteration

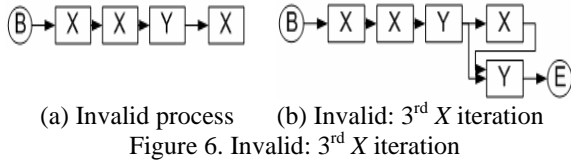
There are five choices to make based on the linkage options detailed above: Begin activity (two options), first activity iteration X and Y (four options each), and final activity iteration X and Y (two options each). The combinations yield 128 potential processes, which include invalid processes.

Many potential Run Time processes are invalid due to the combination $(XY + YX)$. This choice, indicated by $\{ '*', '*', '* XY, YX \}$, appears in quarter of the cases. However, the process may end, and might be a valid process, before reaching these choices, (e.g. $\{ Bs, XE, YE, '*', '* \}$), or might be invalid before reaching these choices, thus the actual number of invalid processes due to this choice is sixteen. Further sixteen invalid cases are described by the process path $B-X-E$, using the selection $\{ BX, XE, '*', '*', '* \}$. These sixteen cases include four cases of $(XY + YX)$, and are counted as part of the $B-X-E$ combination since the process does not reach the choices $(XY + YX)$. Eight invalid options, with 3 executions of X are included in the process path $B-X-X-E$: $\{ BX, XX, '*', XE, '* \}$.

Additional four invalid cases are depicted in Figure 6. Two cases refer to the selection $\{ BX, XX, XY, YX, '* \}$. These two cases are equivalent and yield an invalid process in Figure 6(a). Two additional cases refer to selection $\{ BX, XX, XY, Ys, '* \}$, in which after X has already iterated, linkages from Y are connected to both X and Y , thus a third iteration of X is required. The

case $\{BX, XX, XY, Ys, YE\}$ could complete as a valid process, Figure 6(b); the case $\{BX, XX, XY, Ys, YX\}$, is invalid as discussed above.

Besides *invalid* options, there are distinct options which result in the same valid Run Time process, thus further reducing the count of distinct valid Run Time processes. For example, the selection $\{BX, XY, YE, '*', '*'\}$, representing four options, is equivalent to one valid process depicted in Figure 4(b). Selecting $\{Bs, XE, YE, '*', '*'\}$, produce the process in Figure 4(d).



Applying business rules BR13.1 or BR 13.2 may add distinct Run Time results. For example, if both iterations of Y exist, then the link XY may result in $X-Y(1)$ or $X-Y(2)$, where $Y(i)$ indicate the iteration number.

The total count is 128 *potential* Run Time cases, minus $(16+16+8+4)$ invalid options, yield 84 valid cases. Out of these there are 28 repeated cases (i.e., equivalent to others). The application of business rules BR13.1 or BR 13.2 adds 18 distinct run time cases. In total there are 74 different Run Time processes, which are the result of applying different combinations of business rules to a DSM with two design activities and maximum of two executions for each activity.

4.2. Logic verification issues

In [13], examples were used to demonstrate the implications of the above mentioned business rules for simulating parallel activities. The utilization of implementation rules (IR 4 to IR 7) was discussed in the context of parallel start of multiple activities and parallel completion. The simulation problems arising if these rules are not applied were presented.

The examples in Figure 7 illustrate the implications of adding self-iterations. A short version of the DPM is used: Begin logic activity (B), and End activity (E) are explicitly presented; Input logic and Output logic activities are assumed implicitly before and after the design activities X and Y , respectively.

In Figure 7(a), X and Y activities have self-iteration linkages. According to Eq. (3), the Input logic of X is $IL \Leftarrow B+X$, i.e., signal from Begin or iteration (from Out logic activity). In a similar manner, the input logic of Y , is $IL \Leftarrow X+Y$. The Output logic of Y is Split-Xor between iteration and the link to End activity (IR 9); other options are invalid. There are two Output

logic options for X : either all iterations should complete before continuing to Y (BR 11.1), or (parallelization case) additional X iterations may execute in parallel to Y (BR 11.2). In the second case, the logic of E requires that the last X iteration will complete, though no link exists from X to E in the DPM. This is required since Y may complete and send signal to E , but a new iteration of Y may later start due to rework from X . In general, requiring completion of all activity iterations is derived from validation requirement 3(a).

In Figure 7(b), the option to “jump” to the End activity on the second (or following) iteration of X was added, i.e. applying BR 12.2. Similar to the previous case, the second iterations X_2 (or following) has two options (BR 11.1 and BR 11.2). E has to wait for X to complete all its iterations as in the previous case.

Case	DPM	Valid Logic	Invalid Logic examples			
(a)	B	0		$X \Rightarrow X \otimes Y;$ <i>(BR 11.1)</i> $X \Rightarrow X+Y;$ <i>(BR 11.2)</i> $Y \Rightarrow Y \otimes E;$ <i>(IR 9)</i> $E \Leftarrow X \bullet Y$	$Y \Rightarrow Y+E;$ $Y \Rightarrow Y \bullet E;$	
	X	1	0.1			
	Y		1			0.1
	E					1
(b)	B	0		<i>(BR 12.2):</i> $X_2 \Rightarrow (X \otimes Y)+E;$ $X_2 \Rightarrow X \otimes Y \otimes E;$ $X_2 \Rightarrow (X+Y) \otimes E;$	$X_2 \Rightarrow (X \otimes Y)+E;$ $X_2 \Rightarrow X+Y+E;$; <i>And more options</i>	
	X	1	0.1			
	Y		1			0.1
	E		1			1
(c)	B	0		$X \Leftarrow B+Y;$ $Y \Rightarrow X \otimes Y \otimes E;$ $Y \Rightarrow (X+Y) \otimes E$	$X \Leftarrow B \bullet Y;$ $Y \Rightarrow X+Y+E;$ $Y \Rightarrow X \bullet Y \bullet E;$	
	X	1	0.1			0.1
	Y		1			0.1
	E					1

Figure 7. Additional linkage options for iteration

Figure 7(c) represents a coupled pair of activities. The input logic of X cannot be Join-And since this will cause a deadlock (X waits for Y , while Y waits for X). The valid output logic options of Y are similar to the second iteration of X_2 in the previous case. The identity of logic rules being applied for feedback (from Y) and for the second iteration (of X) was presented in [6]. However, it reflected only second iteration of X in a coupled case and not due to self-iterations.

5. Discussion and Conclusions

Since self-iterations are not addressed in DSM-based simulation literature, the actual simulations have a reduced scope of potential run time processes. The

richness of run time processes due to self-iterations was demonstrated, establishing the multiple variations of interpreting the DSM linkages to current process plans. The example illustrates the complexity of enumerating the run time cases, even for a very simple case; and demonstrates the complexity of verifications of iterative processes.

Verifying rules for the case of two activities and demonstrating their applicability, do not imply other cases correctness. However, such exercise does elucidate the rudiments of iterative process validation, demonstrating the ability to constructively build correct processes, and directs further research endeavor.

6. Appendix A

The relations between IRs, BRs, and the IL and OL equations is shown in Figure 8. IL is described by Eqs. 3 and 8. The OL of the various cases is formulated in Eqs. 4 to 7.

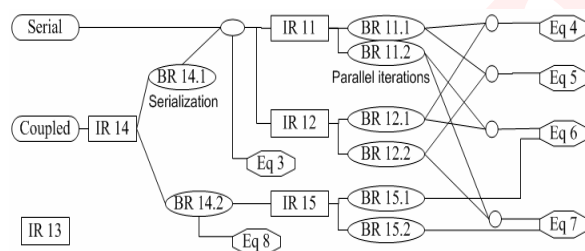


Figure 8. Relations between implementation and business rules

7. References

- [1] Aalst, W.M.P. van der, The Application of Petri Nets to Workflow Management, *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
- [2] Aalst W.M.P. van der, Hee K.M. van, *Workflow Management: Models, Methods, and Systems*, MIT Press, Cambridge, MA. 2002.
- [3] Aalst W.M.P. van der, Hofstede A.H.M. ter, Kiepuszewski B., Barros A.P., Workflow patterns, *Distributed and Parallel Databases* 14(1):5–51, 2003.
- [4] Beck K., *Extreme Programming Explained*, Boston: Addison-Wesley, 2000.
- [5] Browning T.R., Applying the Design Structure Matrix system to decomposition and integration problems: A review and new directions, *IEEE Trans. Eng. Manage.*, 48:292-306, 2001.
- [6] Browning T.R., Eppinger S.D., Modeling impacts of process architecture on cost and schedule risk in product development, *IEEE Trans. Eng. Manage.*, 49(4):428-442, 2002.
- [7] Cho S.H., Eppinger S.D., A simulation-based process model for managing complex design projects, *IEEE Trans. Eng. Manage.*, 52(3): 316-328, 2005.

- [8] Clarkson P.J., Melo A.F., Eckert C.M., Visualization of Routes in Design Process Planning, *In Proceedings of the Fourth International Conference on Information Visualisation (IV2000)*, IEEE Computer Society, pp. 155-164, London, 2000.
- [9] Eppinger SD, Whitney DE, Smith R, Gebala D, A model-based method for organizing tasks in product development, *Res. Eng. Design*, 6(1):1-13, 1994.
- [10] Eppinger SD, Nukala MV, Whitney DE, Generalized models of design iterations using signal flow graph, *Res. Eng. Design*, 9:112-123, 1997.
- [11] Karniel A, Belsky Y, Reich Y, Decomposing the problem of constrained surface fitting in reverse engineering, *Computer-Aided Design*, 37:399-417, 2005.
- [12] Karniel A., Reich Y., From DSM based planning to Design Process Simulation: A review of process scheme verification issues, submitted, 2006(a)
- [13] Karniel A., Reich Y., A Coherent interpretation of DSM plan for PDP simulation, submitted, 2006(b)
- [14] Lévárdu V., Browning T.R., Adaptive test process – designing a project plan that adapts to the state of a project, *Fifteenth Annual International Symposium of the International Council on Systems Engineering (INCOSE)*, 2005.
- [15] Reising W., Rozenberg G., editors, Lectures on Petri Nets I: Basic Models, *Lecture notes in Computer Science* vol 1491, 1998.
- [16] Rinderle S., Reichert, M., Dadam, P., Correctness Criteria for Dynamic Changes in Workflow Systems - A Survey, *Data and Knowledge Engineering*, 50(1):9-34, 2004.
- [17] Sadiq W., Orłowska M. E., Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models, *Lecture Notes in Computer Science*, 1626:195-209, 1999.
- [18] Sadiq S., Orłowska M.E., Sadiq W., Foulger C., Data flow and validation in workflow modeling, *Proceedings of the International Conference in Research and Practice in Information Technology, ADC'2004*, Dunedin, New Zealand, 2004.
- [19] Sered Y., Reich Y., Standardization and modularization driven by minimizing overall process effort, *Computer-Aided Design*, 38(5):405-416, 2006.
- [20] Smith R.P., Eppinger S.D., Identifying controlling features of engineering design iteration, *Management Science*, 43(3):276-293, 1997 (a).
- [21] Smith R.P., Eppinger S.D., A predictive model of sequential iteration in engineering design, *Management Science*, 43(8):1104-1120, 1997(b).
- [22] Steward D.V., *Systems Analysis and Management: Structure, Strategy, and Design*, Petrocelli Books, NJ, 1981.
- [23] Yassine, A., Investigating product development process reliability and robustness using simulation, *Journal of Engineering Design*, in press, 2006.
- [24] Yassine A., Braha D., Complex concurrent engineering and the design structure matrix method, *Concurrent Engineering Research and Applications*, 11(3):165-176, 2003.