INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'07
28 - 31 AUGUST 2007, CITE DES SCIENCES ET DE L'INDUSTRIE, PARIS, FRANCE

COHERENT INTERPRETATION OF DSM PLAN TO PDP SIMULATION

Arie Karniel[1], Yoram Reich[1]
[1]School of Mechanical Engineering, Tel Aviv University

ABSTRACT

Design Processes are highly complex and 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. Planning the Design Process is challenging, and requires simulation tools. The Design Structure Matrix (DSM) is a known method for process planning. Since the DSM itself does not express all the required information for defining the 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. There is a gap in the literature between DSM-based process planning, and process modelling literature concerning the verification of processes. A consistent method of transformation from a DSM model to a logically correct, concurrent process model, in the case of iterative activities, is lacking. Such consistent interpretation of DSM plan to a valid process scheme is proposed in this study. The applicable logic interpretation options are expressed by "Business Rules" that convey different business cases, and can guide automatic translation of the DSM. The proposed framework subsumes logic interpretation types found in the DSM literature.

Keywords: product design process, process planning, process scheme, workflow, design structure matrix, process verification

1 INTRODUCTION

Product Development Process (PDP) can be defined as the entire set of activities required to converting a new concept (market opportunity) into a marketed product. The Design Process (DP), within PDP, reflects the specific design requirements, objectives, and constraints applicable to the product; and is fundamentally iterative. When decisions are made early on, they are based on approximations and uncertain information. As design progresses and knowledge about the product increases, changes emerge. Changes in the design of one component may result in changing the design of other components, thus require iterations. Since iterations are considered as major source of increased product development lead-time and cost [8],[9],[14],[33], performing project activities in an appropriate sequence is critical for minimizing rework due the information interdependencies [21].

The Design Structure Matrix (DSM) [30] is utilized to capture the required product knowledge, and can be used for planning various aspects of the design process [5]. DSM can guide the process planning; yet, trying to model concurrent process logic using DSM is not obvious and may have several interpretations [11],[8]. Furthermore, the DSM does not represent the logic information of the linkages and the presented information is insufficient for implementing process simulations [34],[11].

Karniel and Reich [16] surveyed various simulation process structures and process progress types, with Deterministic, Markov chain, or Monte Carlo methods. The survey analyzed the translation of DSM-based process plan into process simulation in various studies, classified the approaches used, and discussed their strengths and limitations along problems related to process modelling verification.

A gap was identified between the literature concerning the activities sequencing plan based on DSM and the process modelling literature on processes verification. While the need for process verification is emphasizes in the workflow literature [21],[26],[2], in the DSM literature, process logic is defined,

but not verified. Verification check becomes essential for automatic translation of DSM to concurrent process plan model. Such automatic translation is necessary when the product knowledge underlying the DSM has changed or for large-scale processes.
DSM interpretation is non-unique; moreover, different interpretations may be applicable for describing different business processes. However, a consistent method of transformation from a DSM model to a correct concurrent process model, in the case of iterative activities is lacking [16].
The current article focuses on the translation step from DSM-based activity sequencing to a concurrent process plan model denoted as the *Process Scheme*. The interpretation options are define as *Business Rules*. The business rules may derive the *Process Scheme* or manifest in the actual Run Time process. The systematic translation presented, bridges the identified gap between the DSM-based activity sequencing plan phase, and the process structure planning, which is required for actual implementation of design processes. The comprehensive interpretation method presented, explicitly addresses the various business cases previously defined, and some new cases.
The *Process Scheme* is modelled as an activity net, the *DSM net*. In [17], It is proved that the DSM net can be converted to a workflow net (WF-net) [1], which is a specific class of Petri nets [3],[22].

2. LITERATURE SURVEY

2.1. Ontology

The term *Activity* is used to describe a logic step within a process [32]. Activities have pre and post conditions; additionally, they may accept inputs and share outcomes during their performance [11]. In this article, we use the term *Design activity* for indicating performance of design operations (or tasks) in the context of a product component. Typically, these operations are not fully defined. Iteration of a design activity means that design operations are done again in the same context, but not necessarily the same design operations. 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 [16]. For describing the process scheme we use *DSM net*, which is an activity net (with activities as nodes). 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). Construct definitions (Workflow patterns) are elaborated in [4]. The constructs utilized in this study are[1]: 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 interdependencies and process iterations [20],[9]. The DSM representation can be used for various applications [5]; it provides the means to identify serial, parallel and iterative design flows; and model and manipulate iterations and multidirectional information flows [13]. DSM is a square matrix that uses off-diagonal entries to signify the dependency of one element on another. When modelling processes, the lower diagonal portion represents a precedent activity relationship (downstream activities); 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 influence a previously performed activity, which should be performed again, i.e., rework.
Parallel or concurrent activities have no relation linkages; see Figure 1(a). Serial activities are linked by forward link, 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 [34],[33]. In [10] two overlapping activities were modelled as parallel activities with additional lead time and a

[1] The numbers in bracket indicate the definitions in [4], using the terms: Sequence {1}; Parallel split {2}; Multiple choice {6}; Exclusive choice {4}; Synchronization {3}; Multiple merge {8}, and Simple merge {5}. Synchronizing Merge {7} is not used in the current study.

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 [16].

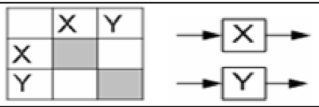
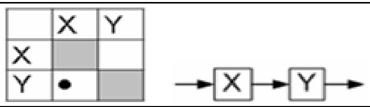
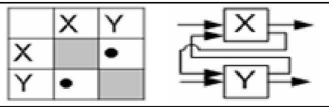
		
(a) Parallel	(b) Serial	(c) Coupled

Figure 1: Activity relations

Different DSM markings are used. Binary DSM uses tick-marks (‘x’) to represent precedence relations [31] (e.g., Figure 1). Numeric DSM represents a measure of the degree of relation or its importance ranking. In Probability DSM [29], linkage probabilities are assigned directly or result from transformation of Numeric (or Binary) DSM [28],[34]. The reordering algorithms: Partitioning, Sequencing (i.e., Partitioning and Tearing), and Clustering typically do not utilize diagonal values [15],[21]. 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 conveying self-iteration probabilities.

DSM-based process planning has several shortcomings: (1) Reordering algorithms that use only the DSM structure are criticized as inadequate indicating that simulation-based optimization is required [5],[8]. (2) The information represented by the DSM (i.e., activity dependencies) is insufficient for simulation, e.g., activity duration, duration due to iteration (learning), cost, and resources are missing. Such information should be specified in addition to the DSM data. (3) As shown in [16], and in the following examples, the DSM information could be interpreted to process logic in several ways.

2.3. Process verification

Process correctness verification is profoundly discussed in the workflow literature [23],[26]. An established formal languages model for process specification is Petri net [22], which is a bipartite directed graph with two node types called *places* and *transitions*, connected by directed *edges* (arcs). Places can contain *tokens*. A transition may ‘fire’, i.e., consume tokens from its input places and add tokens to its output places according to ‘firing rules’. Process activities are modelled 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 modelling and analyzing distributed system behaviour including sequential, conditional, parallel, and iterative routing [2]. WF nets (workflow nets) [1] form a sub-class of 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, 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 open issues; formally: there are no tokens in places other than the termination state. 3) There are no inactive activities that could not execute due to unmet pre-conditions

3. METHOD DESCRIPTION

The planning the design process has two major sub procedures: Creation of the Probability DSM and its reordering; and conversion of the reordered DSM to a process scheme. The Probability DSM creation procedure resembles commonly used procedures (e.g., [28],[35]). The main differences are: the reordering step which is done using the method described in [15]; the use of diagonal elements that indicate self-iteration probability; and the setting of deterministic links (with probability value one), such as link from design activity to testing activity.

3.1 Creating an ordered Probability DSM

The following steps create an ordered probability DSM:

1. Define the design activities.
2. For each design activity, define the parameters influencing other design activities.
3. For each parameter, assign the influence value it may have on design changes and set the influence direction [28].
4. For each link (influence direction), sum the influence values (Impact DSM) [28].
5. Apply DSM reordering (partitioning, clustering and tearing) algorithm in [15].
6. Scale the values to probabilities, linearly [35].
7. Add self-iteration probabilities.
8. Assign deterministic links (i.e., $p=1$).

The first three steps are used by all DSM based procedures, where the value *one* is assigned in Binary DSM. The assignment of other values is done using various considerations [8],[10],[12]. At step four, instead of summing the values, other function could have been used; yet, summation is an appropriate approach, since the resulting values are linearly scaled to probabilities in step six, thus the impact of changing a value is easily traced. In step five, almost any reordering procedure could be used, as long as it keeps the property that a feedback link is necessarily representing an activity cycle. This property is guaranteed for Partitioning [17], and actually for any minimum feedback procedure (otherwise a better DSM reordering could be found, i.e., which minimizes the number of feedback links). Step six could be done before reordering. Linear scaling was demonstrated in [35] to appropriately represent the iteration probabilities. Self-iteration probabilities, is step seven, do not affect the reordering procedure; thus this step could be done earlier as well. 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) [6]). Step eight of assigning deterministic probabilities must be done both after scaling (otherwise the probability will change) and after reordering, since probability $p=1$ cannot be assigned to a feedback link (otherwise the activities will always iterate). The last step, deterministic assignment might be required for activities that have a specific order, i.e., testing should always follow a design activity though it may cause iteration [20].

3.2 Process Validity requirements

Some process validity requirements are derived from the WF nets [1]; additional requirements are specific to design activities. The basic validity requirements for the DSM based process scheme are:

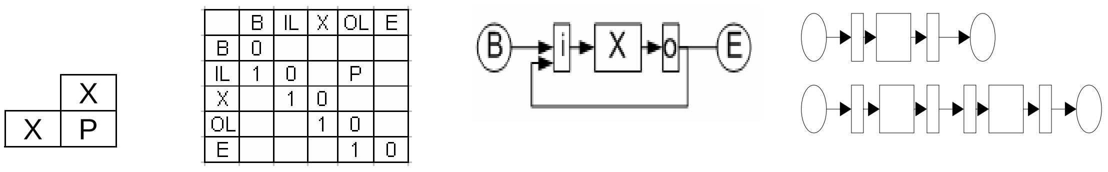
1. The PDP has project characteristics, i.e., should have a defined start and defined end.
2. From any given state, the process should be able to reach the termination state.
3. Reaching the termination state (outcome of executing the End activity) should imply:
   a. all the Design Activities (and all their iterations) have completed; and
   b. each Design 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.

Requirements 1, 2, and 3(a), echo the WF net requirements. Requirement 3(b) implies the completeness of the design process, assuming that all the design activities in the DSM are required. This assumption implies a relation of *what* should be done (assign design activity to product component), rather than specific design operations (i.e., *how* should the design be done). If the process planning was accounting for specific operational tasks, the design activity may have multiple parallel options to choose between (i.e., many ways of performing the design); and then requirement 3(b) could not be justified. The benefit of using 3(b) (with the applicable assumption) is the ability to set a strict simulation termination criterion that can be easily checked. Requirement 4 establishes the need of process records; and may have regulatory implications (e.g., Sarbanes Oxley Act [27]). The last requirement is practical: it is required for simulation, and it follows practical behaviour. In practice, no project has unlimited duration (unlimited resources), thus at some point the process will be enforced to complete or be cancelled.

3.3 Translating ordered DSM into a Process scheme:

Translating an ordered DSM into a Process scheme has the following steps (cf. Figure 2):

1. Translating the DSM into a Design Process Matrix (DPM) by adding logic activities; and defining the logic assigned to the logic activities according to business rules.
2. Converting DPM into an equivalent DSM net, the Current process scheme (C-Process).
3. Using C-Process for simulating a Run Time process scheme (RT-process).



(a) DSM (b) DPM (c) C-Process (d) RT-process

Figure 2: Full process description

The Logic activities added in Step 1 consist of Process Begin and End activities; and Input and Output Logic activities according to Implementation Rules. The interpretation from DPM to the C-Process is structurally straightforward since the process logic is not detailed in the DPM nor in the resulting process scheme (DSM net). Unlike Petri nets, the logic is an additional layer. The benefit of such separation is the ability to change the logic during the RT-process according to the process status (e.g., number of iterations performed). Iteration of design activity is presented as a new activity in the RT-process (i.e., may have different properties than a previous execution). Due to the iterative nature of the activities, many RT-processes could be derived from a single C-process scheme.

4. IMPLEMENTATION RULES:

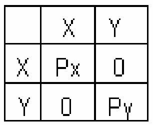
The interpretation of the DSM to a process scheme is not unique, i.e., the presented linkages between activities require additional interpretation for translating the DSM into a valid executable process, which can be simulated. Implementation Rules are used for interpretation. The Implementation Rules (IR) may have sub options 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).

4.1 Single design activity

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=Px*, on the DSM diagonal. Translating a single activity DSM to a DPM implies adding the logic activities: Begin, End, 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 DSM (a) is translated to DPM (b). Logic activities are added: Begin (B), End (E), Input Logic (IL), and Output logic (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 Input logic activity (IL) is Join–Or, i.e., signal from Begin or a feedback signal from the Output logic activity. Respectively, the logic applied to the Output logic activity (decision procedure) is Split-Xor; either sending a feedback signal or a signal to the End activity.
The resulting Current process, in 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. Having at least one input forward link and one output forward link (IR3) satisfy requirement 2, i.e., the process can always complete. Rule (IR3) implies that all activities are on a route from Begin to End, thus finally (after iterations) the process will terminate. A formal explanation is presented in [17]. The number of iterations is practically limited by a threshold on the feedback probability. Setting P_{min} as threshold derives the maximal number of iterations, $N_{\max} = \text{ceil}(\log(P_{\min}) / \log(P))$; where *ceil* is the nearest greater integer. These limitations satisfy requirement 5.

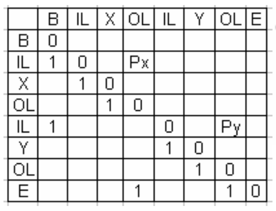
4.2 Parallel independent activities

Two activities may be parallel independent, serial, or coupled. Parallel independent activities process has many similarities to one activity. The X and Y activities can iterate repeatedly, according to the Current process. Figure 3(c) represents the short description version of the process scheme (without indication of In/Out Logic). Results of having one iteration at most, are presented in Figure 3(d).



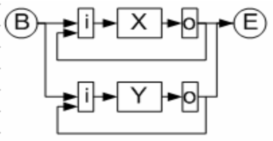
	X	Y
X	Px	0
Y	0	Py

(a) DSM

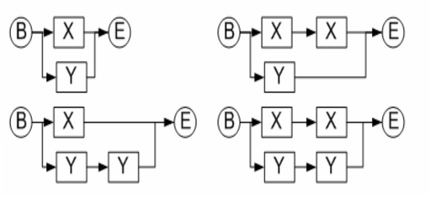


	B	IL	X	OL	IL	Y	OL	E
B	0							
IL	1	0		Px				
X		1	0					
OL			1	0				
IL	1				0		Py	
Y					1	0		
OL						1	0	
E				1			1	0

(b) DPM



(c) C-Process



(d) RT-process

Figure 3: Parallel independent activities

The parallel independent case requires clear definition of the logic assigned to Begin and End activities, as their logic become more functional. The following IRs were introduced as a procedure that handles multiple parallel initiation and parallel termination of multiple paths simulation [16].

(**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.

Both activities accept links from the Begin activity (to IL), having Split-And logic they start in parallel. Both activities are linked to the End activity (from OL), having Join-End logic the End activity will be enabled once both activities have completed performing iterations.

For formulation of the logic being assigned to logic activities, the following symbols are defined:

Logic indication: Join ($\Leftarrow$) for Input logic; Split ($\Rightarrow$) for Output logic.

Logic operations: + (OR); $\bullet$ (AND); $\oplus$ (XOR, Exclusive-Or).

The Join operation implies waiting for signals through the input links, and is equivalent to Boolean logic equation. However, Split operation implies a decision procedure, indicating the activities to which signals should be sent. Split-Xor is an ordered choice procedure, e.g., $A \Rightarrow B \oplus C$, means first check sending signal to *B*, if signal was not sent to *B*, then send signal to *C*. Join-Xor: is not in use in the current work, since the required behaviour is always Join-Or.

The short form of multi-variable logic operations: Multiple-Or $\Sigma(A_i) = A_1 + A_2 + \cdots + A_n$; Multiple-And $\Pi(A_i) = A_1 \bullet A_2 \bullet \cdots \bullet A_n$; and Multiple-Xor $\otimes(A_i) = A_1 \oplus A_2 \oplus \cdots \oplus A_n$, 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 is presented for the implementation of Begin and End:

$$Begin \Rightarrow \Pi(F_i) \tag{1}$$

$$End \Leftarrow \Pi(F_i) \tag{2}$$

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

(**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. Equation 3). The Out logic procedure may send signals to any feedback link, or (exclusively) send signals to all forward links, but not both (cf. Equation 4).

$$IL \Leftarrow \left(\Pi(F_i)\right) + \left(\Sigma(I_i)\right) \tag{3}$$

$$OL \Rightarrow \left(\Sigma(I_i)\right) \oplus \left(\Pi(F_i)\right) \tag{4}$$

4.3 Serial activities

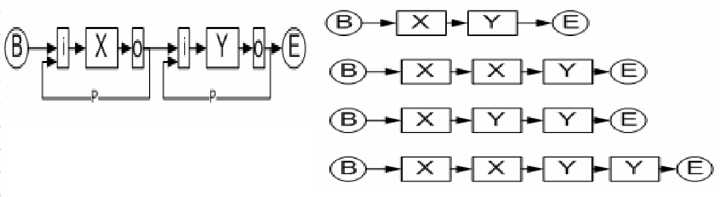
Serial activities in a design process might be the result of standardization [28], e.g., while the design of a standard part A may influence the design of B, changes in the design of B will not affect the design of A (otherwise it is not standard). Figure 4 describes an example of translation stages from DSM to RT-process. The link from activity *X* to activity *Y* has probability *Pxy* at Figure 4(a). It is translated to a link from Out Logic activity (OL) of *X* to Input Logic activity (IL) of *Y* at Figure 4(b).

	X	Y
X	Px	0
Y	Pxy	Py

(a) DSM

	B	IL	X	OL	IL	Y	OL	E
B	0							
IL	1	0		Px				
X		1	0					
OL			1	0				
IL				Pxy	0		Py	
Y					1	0		
OL						1	0	
E							1	0

(b) DPM



(c) C-Process (d) RT-process

Figure 4: Serial activities

The validity requirement 3b (Section 3), indicating one activity 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 (next activity and End activity) hinders IR 9, and was shown in [16] to cause invalid processes. The interpretation of the forward link with probability *Pxy* in case of iterations is not unique. The issues to be considered: If *X* iterates, can *Y* start in parallel, or should it wait until *X* completes all its iterations. The interpretation may be one of the following Business Rules:

(**IR 11**): Output logic options: sending completion signal(s) to following serial design activities may follow one of the business rules:

(**BR 11.1**): Sending only once the activity has completed all its iterations.

(**BR 11.2**): Sending once the activity has completed its first execution (i.e., early start of next activity); yet, sending a signal to End activity can be done only once all iterations have completed (i.e., IR 9).

The first case (business rule BR 11.1) is a serial process with self- iterations, depicted in Figure 4(c). Since activity *Y* may start only after the completion of all iterations of activity *X*, and iterations are serial, the whole process is serial, Figure 4(d). In this case, the probability *p=Pxy* has no simulative meaning. It only indicates a forward process link, thus it could be translated to probability *p=1*. This rule is the default rule and it complies with the basic Out-logic rule presented in Equation (4).

4.4 Serial activities with parallel execution

In the latter case (BR 11.2), there might be iterations of the previous activity *X* that are executed in parallel to activity *Y*. The implementation of this case requires additional rules, which are useful for other parallel cases as well. Such possibility is not studied in existing literature.

(**IR 12**): Output logic: signal to End Activity. Second (or later) execution of an activity (e.g., *X*), may send signal to End Activity, while the following serial activities (e.g., *Y*) have started execution (or have completed):

(**BR 12.1**): On second (or later) execution, the activity must be followed by its next serial activities.

(**BR 12.2**): On second (or later) execution of the activity, the next activities may follow, or the End activity may follow (not both, subject to IR 9).

Four business rule combinations change the Output logic, and are manifested in the DPM. The first was the case of BR 11.1 and BR 12.1, already described by Equation (4). The combination BR 11.1 and BR 12.2 is formulated in Equation (5); BR 11.2 and BR 12.1 in Equation (6); and BR 11.2 with BR 12.1 in Equation (7). See appendix A

$$OL \Rightarrow (\Sigma(\mathrm{Ii}) \oplus \Sigma(\mathrm{Fi})) \oplus End \qquad (5)$$

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

$$OL \Rightarrow (\Sigma(\mathrm{Ii}) + \Sigma\,\mathrm{Fi}) \oplus End \qquad (7)$$

Allowing early start option (BR 11.2), is deriving Run Time options (IR 13), which are not described by the PDM. These options can be addressed only in the RT-process scheme.
(**IR 13**): Iterations of same activity cannot occur in parallel:
(**BR 13.1**): While the activity is executing, input link signals are directed to this activity (i.e., to its input Logic activity).
(**BR 13.2**): While the activity is executing, input link signals are directed to the next iteration of the activity, (next iteration cannot start until current iteration has completed).

Rule (IR 13) has validation implications; avoiding activities proliferation in simulation. In general, it is assumed the same resource is doing additional iterations of the same design activity (this assumption can be used do decrease the duration of iterative activities, due to learning). If two resources perform the design of the same component (e.g., developing several concepts in parallel), then these activities should be defined distinctly (i.e., not iterations of same activity). In the context of administrative processes, the options described above were defined as Lack of Synchronization conflict [25]; i.e., multiple requests for same activity. The first option (BR 13.1) entails defining the consequences of the additional iteration rework. Typically, the activity duration will just increase. Recalculating the activity duration is similar to the overlapping [10], and can be addressed by the same approaches.

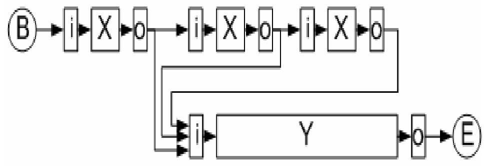
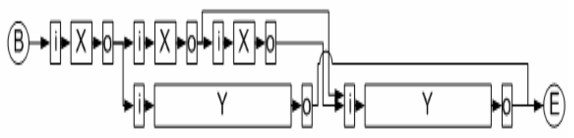
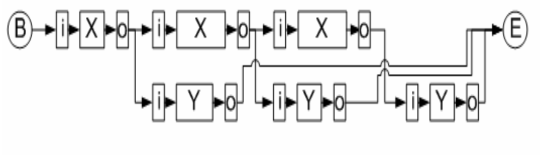
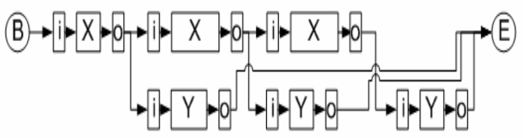
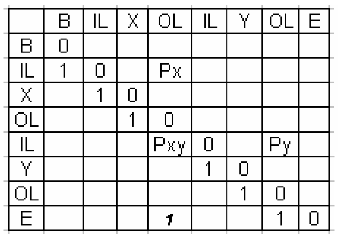
	
(a) RT: BR 13.1 $d(x) < 0.5 \cdot d(y)$	(b) RT: BR 13.2, $d(x) < 0.5 \cdot d(y)$
	
(c) RT: BR 13.1, $d(x) > d(y)$	(d) RT: BR 13.2, $d(x) > d(y)$

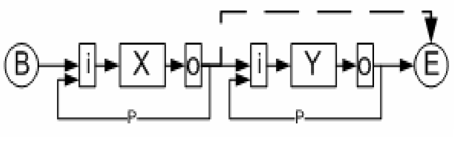
Figure 5: Serial activities Serial Activities RT-process BR 11.2 + BR 12.1

The implications of applying the rules are dependent on the activities relative duration. The case $duration(X) > duration(Y)$, yields similar Run Time processes for both business rules, Figure 5(c) and (d), respectively. The more interesting case $duration(X) < 0.5 * duration(Y)$ is depicted in Figure 5(a) and (b). The relative length of the activity box, graphically presents the relative duration. This presentation is not precise, since Logic activities duration is zero. Yet, the Run Time process is presented as if it was sketched over time axis. The exhibited difference in Run Time process, for self-iterations, due to differences in activities duration, is a new result, not mentioned in the literature.

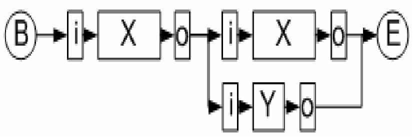


	B	IL	X	OL	IL	Y	OL	E
B	0							
IL	1	0		Px				
X		1	0					
OL			1	0				
IL				Pxy	0		Py	
Y					1	0		
OL						1	0	
E				1			1	0

(a) DPM



(b) C-Process



(c) RT-Process

Figure 6: Serial activities BR 11.2 + BR 12.2

The DPM structure and the Current process scheme, after adding the link to the End activity, (at second execution or later), are depicted in Figure 6(a) and (b), respectively. The main structural difference is the additional link from OL (Out Logic) activity following *X* to the End activity (applicable only to the second iteration of *X* activity). The dashed line in the C-process indicates, respectively, that the logic is applicable in the second execution (or later). The process cannot end after the first execution of *X* without performing activity *Y* (validation rule 3(b)). This option is always economic in terms of time; yet, quality may degrade, since the implications of the change (in *X* design activity on *Y* design activity) are not re-checked. This option is more general than previous one, as all the RT-processes generated in the previous case (BR 11.2 and BR 12.1) could be generated by this case, but not vice versa. An example of a different Run Time process is depicted in Figure 6(c).

4.5 Coupled activities

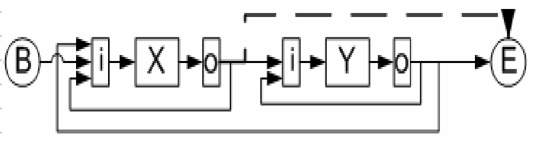
The coupled activities case may have serialization requirements (e.g., testing activity should serially follow design activities).
(**IR 14**): Coupled activity execution start:
(**BR 14.1**) (serialization): coupled activity may start, after its previous activity (according to DSM) has completed at least once.
(**BR 14.2**) (parallel): coupled activity may start in parallel to all the other activities in the same activity loop.
Serialization (BR 14.1) logic is described by the same four logic cases (See appendix A). The link *Pyx* is assigned from the OL of *Y* activity to the IL of *X* activity in the DPM Figure 7(b). Serialization is defined by setting *Pxy=1,* i.e., *Y* must follow *X*. The case of applying BR 12.2 (ability to send signal to End after second execution, or to next activities) is depicted in Figure 7(b) and (c).

	X	Y
X	Px	Pyx
Y	Pxy	Py

(a) DSM

	B	IL	X	OL	IL	Y	OL	E
B	0							
IL	1	0		Px			Pyx	
X		1	0					
OL			1	0				
IL				Pxy	0		Py	
Y					1	0		
OL						1	0	
E				1			1	0

(b) DPM BR 14.1



(c) C-Process BR 12.2

Figure 7: Serialized Coupled activities, BR 14.1

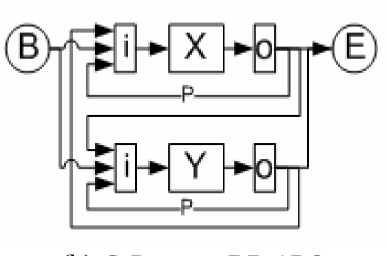
Starting coupled activities concurrently (BR 14.2) results in processes whose Output logic is similar to the case of BR 11.2 (parallelization of the iterations of serial activities); but with different Input logic. The Input logic for BR 14.2 is formulated in Equations (8), (replacing equation (3)).

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

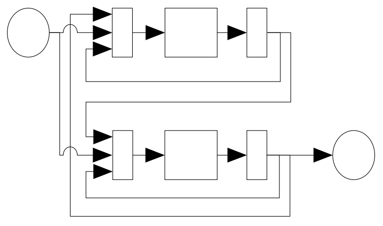
The difference in implemented by setting a link with probability *p=1* from Begin, Figure 8(a).

	B	IL	X	OL	IL	Y	OL	E
B	0							
IL	1	0		Px			Pyx	
X		1	0					
OL			1	0				
IL	1			Pxy	0		Py	
Y					1	0		
OL						1	0	
E				1			1	0

(a) DPM BR 14.2 + BR 15.2



(b) C-Process BR 15.2



(c) C-Process BR 15.1

Figure 8: Parallel Coupled activities, BR 14.2

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.
The DPM in Figure 8(a) was assigned with rule BR 15.2, i.e., there is a link from OL of *X* to End (after second execution), and the resulting C-process is depicted in (b). If BR 15.1 was used, the latter link was not included and the resulting C-process would be as in (c). Additional implications of utilizing self-iterations were further studied in [19] and provided additional insight to iterative processes modelling.

4.6 Logic verification issues

The DSM (and DPM) does not fully define the process logic, and can not represent both input logic and output logic in the general case [16]. Setting logic as additional information, by using additional matrices, using rules, or using logic formulation, is not trivial. Not any set of rules may apply. The latter assertion is demonstrated by the following examples that provide more insight to iterative processes simulation logic and implementation. Note: the input and output logic activities are implicit (not presented), before and after the design activities *X* and *Y*, respectively.

case	DPM	Valid Logic	Invalid Logic examples
(a)	$\begin{array}{ccccc} & B & X & Y & E \\ B & 0 & & & \\ X & 1 & 0.1 & & \\ Y & 1 & & 0.1 & \\ E & & 1 & 1 & 0 \end{array}$	$B \Rightarrow X \bullet Y$; $E \Leftarrow X \bullet Y$; $X \Rightarrow X \oplus E$; $Y \Rightarrow Y \oplus E$;	$B \Rightarrow X+Y$; $B \Rightarrow X \oplus Y$; $E \Leftarrow X+Y$; $X \Rightarrow X+E$; $X \Rightarrow X \bullet E$; $Y \Rightarrow Y+E$; $Y \Rightarrow Y \bullet E$;
(b)	$\begin{array}{ccccc} & B & X & Y & E \\ B & 0 & & & \\ X & 1 & 0.0 & & \\ Y & 1 & 1 & 0.0 & \\ E & & & 1 & 0 \end{array}$	$B \Rightarrow X \bullet Y$; $Y \Leftarrow B \bullet X$;	$Y \Leftarrow B+X$;
(c)	$\begin{array}{ccccc} & B & X & Y & E \\ B & 0 & & & \\ X & 1 & 0.1 & & \\ Y & 1 & 1 & 0.1 & \\ E & & & 1 & 0 \end{array}$	$X \Rightarrow X \oplus Y$; $X \Rightarrow X+Y$; ($E \Leftarrow X \bullet Y$)	
(d)	$\begin{array}{ccccc} & B & X & Y & E \\ B & 0 & & & \\ X & 1 & 0.1 & 0.1 & \\ Y & 1 & 1 & 0.1 & \\ E & & \mathit{1} & 1 & 0 \end{array}$	$X \Leftarrow B+Y$; $X \Rightarrow X \oplus Y$; $X_2 \Rightarrow X \oplus Y \oplus E$; $X_2 \Rightarrow (X+Y) \oplus E$; $Y \Leftarrow B+X+Y$; $Y \Rightarrow (X+Y) \oplus E$;	$X \Rightarrow (X \oplus Y)+E$; $X \Rightarrow X+Y+E$; $X \Rightarrow Y \bullet E$; and more

Figure 9: Process logic examples

Parallel activities example is depicted in Figure 9(a). The option of using Split-Or logic for Begin and Join-Or logic for End, (***B⇒X+Y*** and ***E⇐X+Y***) respectively, is legitimate in a general process. However, in the context of Design process it is invalid, since we assume that every activity must be performed at least once. The logic ***B⇒X⊕Y*** will always cause one of the activities not to start. In general, the output logic of forward links, being performed first time, should always be Split-And (to ensure that next activities execute at least once). Having self-iterations prevents using output logic other than Split-Xor between the link to End and other links, as otherwise the process might terminate before all design activity iterations have completed. A serial case without self-iterations is depicted in Figure 9(b). The link from Begin to *Y* is not required according to the DPM generation rules; and is added to demonstrate the input logic of *Y*. The input logic must be Join-And of all forward links (i.e., *Y* should wait to all previous activities to complete), as it is serial.
The output logic of *X* is considered once adding self-iterations; depicted in (c). There are two options: either all *X* iterations should complete (***X⇒X⊕Y***), or *Y* might start in parallel to an iteration of *X* (Note: the case ***X⇒X•Y*** is a sub-option, i.e., *Y* must start with every iteration of *X*). Since all iterations should terminate the End activity should wait for all *X* iterations to complete, though there is no indication of a link from *X* to End. Without such logic the process may end once *Y* completes, while *X* is still iterating (and may cause additional iterations of *Y*). The analysis is quiet similar to the examples given in [3]. Yet, waiting for activity iteration with no indication of link to End activity, will be regarded as flawed situation in a simple Petri net; and can be solved in high level Petri net by adding a data link.
The case of coupled activities with parallel start and early termination is depicted in (d). The input logic of *X* is (typically) defined as Join-Or of forward and feedback links. In this case, Xor and And options are also applicable since *Y* starts in parallel (otherwise, Join-And would cause a deadlock). Similarly, the input logic of *Y* is Join-Or (having multiple activities, the forward links would have Join-And logic, equation 8). The output logic of *X* does not allow link to End at first execution, but only after iterations (i.e., from second execution X_2).

5. DISCUSSION AND CONCLUSIONS

The transformation of a DSM plan to a process scheme is not unique, and could easily lead to inapplicable process logic options that yield invalid processes. Therefore, business rules are required for choosing between the applicable options, particularly if the transformation to process plan is to be done automatically or in a changing knowledge environment. This article presented the application of a basic set of business rules for transforming an ordered DSM to DPM. The application of implementation rules IR 1 to IR 7, which are used for this transformation, results in a net that satisfies basic process validation requirements (formal proof are presented in [17]). The application of IR 11 to IR 15 business rule options can be predefined; or monitored by applying additional business rules, according to the process state, due dates, or other considerations. Process logic and other process parameters (e.g., cost and resources) that complement DSM information have to be represented outside the DSM; together they provide the foundation for process planning based on simulations. The current study provides support for establishing a dynamically changing process planning mechanism, entitled Dynamic Product Development Process [18].

APPENDIX A

The relations between Implementation Rules, Business Rules, Input and Output Logic equations, are shown in Figure 10. Input logic is described by equations 3 (for serial or serialized) and 8 (coupled activities), respectively. The Output logic of the various cases is formulated in equations 4 to 7.

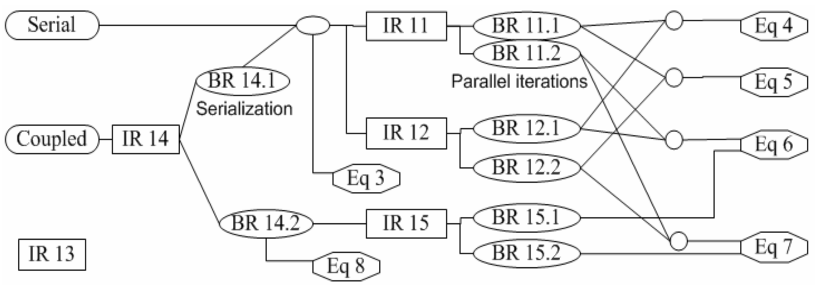


Figure 10: Relations between implementation and business rules

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, Exterminating the dynamic change bug: A concrete approach to support workflow change, *Information Systems Frontiers*, 3(3):297-317, 2001.

[3] Aalst W.M.P. van der, and Hee K.M. van, *Workflow Management: Models, Methods, and Systems*, MIT Press, Cambridge, MA. 2002.

[4] 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.

[5] Abdelsalam H.M.E., and Bao H.P., A simulation-based optimization framework for product development cycle time reduction, *IEEE Trans. on Eng, Management*, 53(1):69-85, 2006.

[6] Beck K., *Extreme Programming Explained*, Boston: Addison-Wesley, 2000.

[7] Browning T.R., Applying the Design Structure Matrix system to decomposition and integration problems: A review and new directions, *IEEE Trans. on Eng. Management*, 48:292-306, 2001.

[8] Browning T.R., and Eppinger S.D., Modeling impacts of process architecture on cost and schedule risk in product development, *IEEE Trans. on Eng. Management*, 49(4):428-442, 2002.

[9] Browning T.R., Fricke E., and Negele H., Key concepts in modeling product development processes, *Systems Engineering*, 9(2):104-128, 2006.

[10] Cho S.H., Eppinger S.D., A simulation-based process model for managing complex design Projects, *IEEE Trans. on Eng. Management*, 52(3):316-328, 2005.

[11] Clarkson P.J. and Hamilton J.R., 'Signposting', a parameter-driven task-based model of the design process, *Research in Engineering Design*, 12(1):18-38, 2000.

[12] Eckert C.M., Keller R., Earl C., Clarkson P.J., Supporting change processes in design: Complexity, prediction and reliability, *Reliability Eng. and System Safety*, 91(12):1521-1534, 2006.

[13] Eppinger S.D., Nukala M.V., and Whitney D.E., Generalized models of design iterations using signal flow graph, *Research in Engineering Design*, 9:112-123, 1997.
[14] Eppinger S.D., Whitney D.E., Smith R., and Gebala D., A Model-based method for organizing tasks in product development, *Research in Engineering Design*, 6(1):1-13, 1994.
[15] Karniel A., Belsky Y., Reich Y., Decomposing the problem of constrained surface fitting in reverse engineering, *Computer-Aided Design*, 37: 399-417, 2005.
[16] Karniel A., Reich Y., From DSM based planning to design process simulation: A review of process scheme verification issues, *submitted*, 2006a.
[17] Karniel A., Reich Y., DSM based implementation of process schemes, *submitted*, 2006b.
[18] Karniel A., Reich Y., Karniel A., Reich Y.: Managing Dynamic New Product Development Processes. ***submitted***, (2006c)
[19] Karniel A., Reich Y.: Simulating Design Processes with self-iteration activities based on DSM planning. in proceedings of the International Conference on Systems Engineering and Modeling - ICSEM'07. Haifa (2007)
[20] Lévárdy 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)*, 10 July to 15 July, 2005.
[21] Meier C., Yassine A., Browning T., *Design Process Sequencing with Competent Genetic Algorithms*, PDRL working paper # PDL-2006-03, March, 2006.
[22] Reising W., Rozenberg G., editors, *Lectures on Petri Nets I: Basic Models*, Lecture notes in Computer Science, vol 1491, 1998.
[23] 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(a).
[24] Rinderle, S., Reichert, M., Dadam, P., Flexible support of team processes by adaptive workflow systems, *Distributed and Parallel Databases*, 16(1): 91-116, 2004(b).
[25] Sadiq W. and Orlowska M. E., Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models, Lecture Notes in Computer Science, 1626:195-209, 1999.
[26] Sadiq S., Orlowska 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.
[27] Sarbanes P. (S. 2673), Oxley M.G (H.R. 3763), *Public Company Accounting Reform and Investor Protection Act of 2002*, ("Sarbanes Oxley Act"), *Pub. L. No. 107-204, 116 Stat. 745*, July 30, 2002. http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=107_cong_bills&docid=f:h3763enr.tst.pdf
[28] Sered Y., and Reich Y., Standardization and modularization driven by minimizing overall process effort, *Computer-Aided Design*, 38(5):405-416, 2006.
[29] Smith R.P. and Eppinger S.D., A Predictive model of sequential iteration in engineering design, *Management Science*, 43(8): 1104-1120, 1997
[30] Steward D.V., *Systems Analysis and Management: Structure, Strategy, and Design*, Petrocelli Books, NJ, 1981(a).
[31] Steward D. V., The design structure system: a method for managing the design of complex systems, *IEEE Trans. on Eng. Management*, 28:71-74, 1981(b).
[32] WFMC, *Workflow Management Coalition Terminology and Glossary*, Technical report WFMC-TC-1011, issue 3, Workflow Management Coalition, 1999.
[33] Whitfield R.I., Duffy A.H.B., Gartzia-Etxabe L. K., Identifying and evaluating parallel design activities using the Design Structure Matrix, *International Conference on Engineering Design, ICED 05*, Melbourne, August 15-18, 2005.
[34] Yassine A., Braha D., Complex concurrent engineering and the Design Structure Matrix method, *Concurrent Engineering Research and Applications*, 11(3):165-176, 2003.
[35] Yassine A., Investigating product development process reliability and robustness using simulation, *Journal of Engineering Design*, 2006.

Contact: Y. Reich
Tel Aviv University, School of Mechanical Engineering, Tel Aviv 69978, Israel
Phone: +972-3-6407385, Fax: +972-3-6407617
e-mail: yoram@eng.tau.ac.il , URL: http://www.eng.tau.ac.il/~yoram