WIT Modeling Techniques

June 3, 2008

Bob Wittrock

IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Introduction

WIT's data objects and attributes explicitly model various aspects of the manufacturing planning problem we call "implosion". But there are many aspects of problems that can be solved by WIT that are not explicitly part of WIT's model. To model these aspects using WIT's data objects requires additional modeling techniques implemented in the application program. This document is intended to be an informal collection of some of the modeling techniques we have thought of and believe to be useful.

As of this writing, this document discusses techniques for modeling the following problem characteristics:

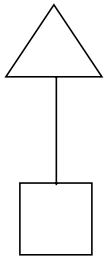
1. Lower Bounds on Capacity Usage
2. Solution Targets
3. Production Smoothing
4. Setups
5. Variable Length Periods

The techniques are presented in order of (what is thought to be) increasing model complexity. But we begin by describing a unifying concept that is at the heart of most of these techniques.

Regulators

Before discussing modeling techniques that address specific issues, it will be helpful to define a generally useful modeling structure, which might be called a “regulator”. A regulator is simply an operation and a capacity connected by a BOP entry:

FIGURE 1 Regulator



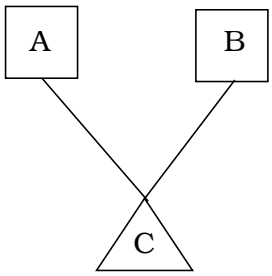
A regulator is typically used by attaching it to existing objects in the model with BOM entries and/or BOP entries. Once attached, the regulator allows additional controls to be applied to the existing model by setting the attributes of the objects that make up the regulator itself.

As we shall see, this stucture turns out to be very useful. As of this writing, all of the modeling techniques discussed in this document that are structural in nature make use of the regulator structure.

Lower Bounds on Capacity Usage

Capacities normally function as a upper bound kind of constraint: an operation cannot be executed so much that the supplyVol of a capacity is exceeded. But sometimes one needs to impose a lower bound on the usage of a capacity. Consider the following example:

FIGURE 2 Lower Bounds on Capacity Usage: Example Problem

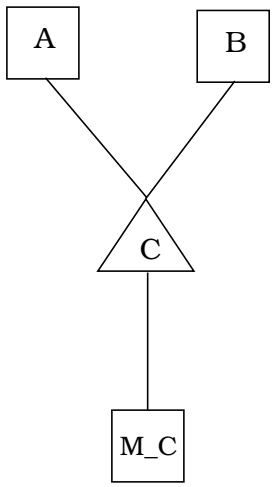


We would like to impose a lower bound on the consVol of part C.

WIT Model

Since this technique uses bounds, it requires optimizing implosion. The technique is to replace the capacity with a regulator, as indicated in the following example:

FIGURE 3 Lower Bounds on Capacity Usage: Example of Technique



- If objchoice = 1, set C.obj1ScrapCost to a very large number. If objchoice = 2, set C.unitCost to a very large number. The effect of this is to prohibit scrapping of C.
- Next set C.supplyVol = 0. It follows that, for each period, t:
  C.consVol[t] = M_C.execVol[t]
- Then set the hard upper bound on M_C.execVol to the actual supplyVol of capacity C. This will cause capacity C to be used no more than its actual supplyVol.
- Finally set the soft or hard lower bound on M_C.execVol to the desired lower bound. The structure will then cause these bounds to be applied to C.consVol.

Solution Targets

Sometimes the user has a pre-specified "ideal" solution to the implosion problem and wants to find a feasible solution as close as possible to the ideal. We might call the pre-specified solution a "target" solution. Since the target solution might not be feasible, one would specify penalties for deviations from it. The targeting concept could be used in conjuction with ordinary implosion: one might use an ordinary objective function combined with penalties for deviations from the target solution. One case where this might arise is if a series of implosions is being performed on similar problems, with only a little data being changed from one implosion to the next. In such a siutation, it might be undesirable for the solution to change drastically when the problem changes slightly and so the target would be the solution to the previous implosion.

An implosion solution consists of three primary components:

- The execVol for each operation.
- The shipVol for each demand.
- The subVol for each substitute.

In some cases, targeting may only be desired for one or two of these components. We will present a targeting technique for each component separately. The techniques apply only to optimizing implosion and require objective #1.

Problem Definition/Assumptions

The execution target solution is given as:

- theOperation.execVolTar[t]

for each operation and period, t.

The penalties are specified as linear functions of the positive and negative deviations from the targets. The penalty rates are given as:

- theOperation.execHiPen[t]
- theOperation.execLoPen[t]

For one operation in one period, the penalty is:

op.execHiPen[t] * [op.execVol[t] - op.execVolTar[t]]$^{+}$ +

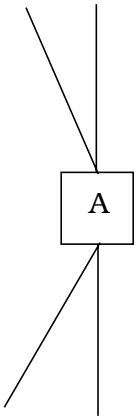
op.execLoPen[t] * [op.execVol[t] - op.execVolTar[t]]$^{-}$

The shipment targets and substitution targets and their penalties would be specified similarly.

Execution Targets: WIT Model

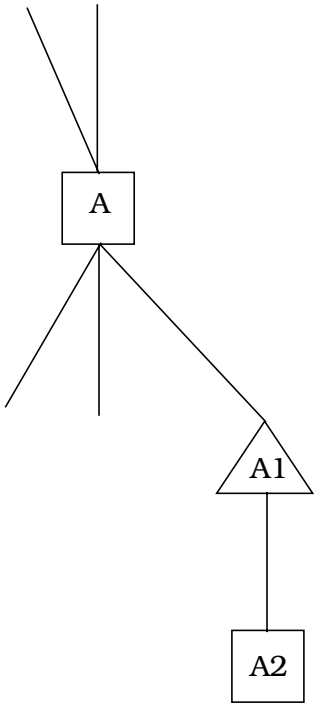
Consider the following example:

FIGURE 4 Execution Targets: Example Problem



To apply the execution targeting technique, attach a regulator to the operation with a BOM entry, as follows:

FIGURE 5 Execution Targets: Example of Technique



For each period, t, set:

- A1.supplyVol[t] = A.execVolTar[t]
- A2.obj1ExecCost[t] = A.execHiPen[t]
- A1.obj1ScrapCost[t] = A.execLoPen[t]

This structure will cause the following relationships to hold for each period, t:

$A2.obj1ExecVol[t] = [op.execVol[t] - op.execVolTar[t]]^{+}$

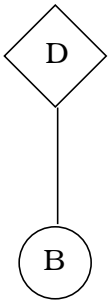
$A1.obj1ScrapVol[t] = [op.execVol[t] - op.execVolTar[t]]^{-}$

And this, in turn, imposes the specified penalties on deviations from execVolTar.

Shipment Targets: WIT Model

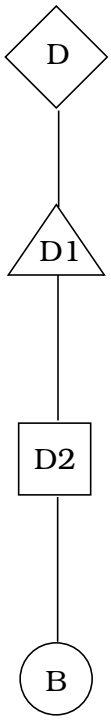
Consider the following example:

FIGURE 6 Shipment Targets: Example Problem



To apply the shipment targeting technique, first transform this problem into a execution target problem. To do this, attach a regulator to the part with a BOM entry and then attach the demand to the regulator, as follows:

FIGURE 7 Shipment Targets: Example Problem Tranformed



- For each period, set D1.obj1ScrapVol[t] to a very large number. This causes the following relationship to hold for each period, t:

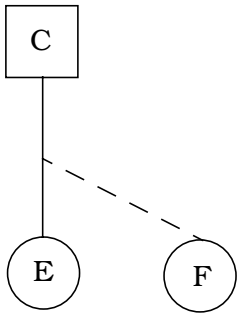
  D2.execVol[t] = D.shipVol[t]

- Then apply the execution target technique to operation D2, using the shipment targets as execution targets and shipment deviation penalties as execution deviation penalties. The resulting structure will then impose these penalties on deviations between D.shipVol and its target.

Substitution Targets: WIT Model

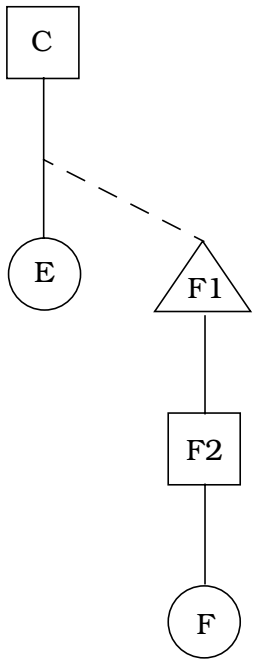
Consider the following example:

FIGURE 8 Substitution Targets: Example Problem



To apply the substitution targeting technique, first transform this problem into an execution target problem. To do this, attach a regulator to the part consumed by the substitute with a BOM entry and then attach the substitute to the regulator, as follows:

FIGURE 9 Substitution Targets: Example Problem Transformed



- The various attributes for the original substitute BOM entry should be applied to the BOM entry connecting the substituted part (F) with with added operation (F2): usageRate, offset, etc. The attributes of the substitute BOM entry itself should be set to default values.
- For each period, t, set F1.obj1ScrapVol[t] to a very large number. This causes the following relationship to hold for each period, t:

  F2.execVol[t] = subEntry.subVol[t]
- Then apply the execution target technique to operation F2, using the substitution targets as execution targets and substitution deviation penalties as execution deviation penalties. The resulting structure will then impose these penalties on deviations between subEntry.subVol and its target.

Production Smoothing

Sometimes the user desires an implosion solution whose prodVols do not change much from one period to the next. This is called production smoothing. Since, in WIT, the prodVols are derived linearly from the execVols, this is really a requirement for execution smoothing and we will present a modeling technique for this. The technique only applies to optimizing implosion and requires objective #1.

Problem Definition/Assumptions

The smoothing requirement is specified as a linear penalty of the absolute difference between the execVol in one period and in the previious period. The penalty rates are given as:

theOperation.execDevPen[t]

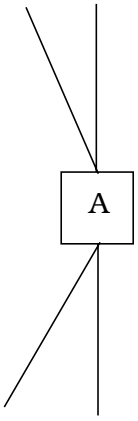
For one operation in one period, t > 0, the penalty is:

op.execDevPen[t] * | op.execVol[t] - op.execVol[t-1] |

WIT Model

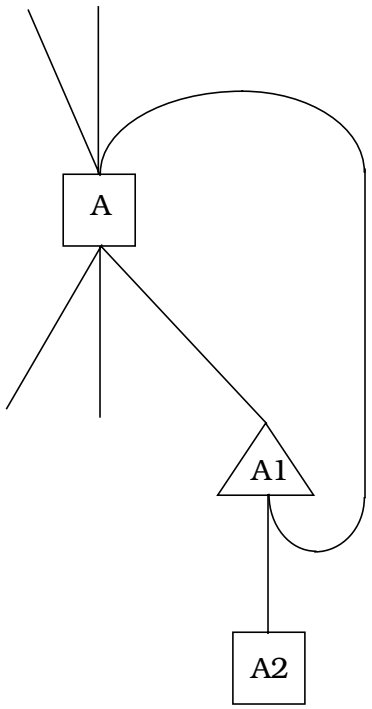
Consider the following example:

FIGURE 10 Execution Smoothing: Example Problem



To apply the execution smoothing technique, attach a regulator to the operation with a BOM entry and a BOP entry, as follows:

FIGURE 11 Execution Smoothing: Example of Technique



For each period t, set:

- A1.objScrapCost[t] = A.execDevPen[t]
- A2.obj1ExecCost[t] = A.execDevPen[t], if $t \geq 0$
- BOP entry (A, A1).offset[t] = -1.0

Also, set

- A2.obj1ExecCost[0] = 0.0
- BOP entry (A, A1).expAllowed = FALSE
  (to avoid an explodeable cycle).
- BOP entry (A, A1).latestPeriod = nPeriods - 2

Setting latestPeriod to nPeriods - 2 turns off the BOP entry in the last period. The BOP entry is not needed in the last period, and if it one didn't turn it off, its offset would imply production after the

end of the time horizon, which would prevent execution of operation A in the last period.

To see the effect of this structure, first consider A.execVol[0]:

A.execVol[0] = A1.consVol[0]

In period 0, there is no reason to scrap A1 and incur the penalty, so

A1.scrapVol[0] = 0

Thus

A1.consVol[0] = A1.prodVol[0]

Due to the offset on BOP entry (A, A1), A1 can only be produced in period 0 by operation A2, so

A1.prodVol[0] = A2.execVol[0]

Thus

A.execVol[0] = A2.execVol[0]

which does not incur a penalty.

Now consider A.execVol[t], for some period, t > 0:

A.execVol[t] = A1.consVol[t]

and

A1.consVol[t] = A1.prodVol[t] - A1.scrapVol[t]

and

A1.prodVol[t] = A2.execVol[t] + A.execVol[t-1]

Thus

A.execVol[t] - A.execVol[t-1] = A2.execVol[t] - A1.scrapVol[t]

Thus:

- If A.execVol[t] > A.execVol[t-1], the difference is A2.execVol[t], which is penalized at the rate of A.execDevPen[t].
- If A.execVol[t] < A.execVol[t-1], the difference is A1.scrapVol[t], which is penalized at the rate of A.execDevPen[t].
- If A.execVol[t] = A.execVol[t-1], no penalty is incurred.

And so the smoothing deviation penalties are applied as desired.

Finally, it may be desireable to smooth other aspects of the solution besides execVol, i.e., shipVol and subVol. This can be done by transforming the smoothing problem in question into an execution smoothing problem by adding the appropriate regulators, as was done for solution targeting.

Setups

As of this writing, we know of three ways to model setups in WIT:

1. Constant Estimate
2. Linear Estimate
3. Once-Per-Family

Constant Estimate

Problem Definition/Assumptions

The period length is long enough that multiple occurances of the same setup happen during a period and you can estimate in advance the amount of setup time that will be incurred by a capacity. This might apply if the setup time is independent of the operation being performed and there is a fixed lot size.

WIT Model

This model is compatible with all WIT solution functions.

For each capacity that has setup times, estimate the total setup time that will be incurred by the capacity in each period, and decrement the supplyVol of the capacity by that much.

Linear Estimate

Problem Definition/Assumptions

The period length is long enough that the setup time incurred by each BOM entry can be estimated as proportational to the execution volume the consuming operation. For example, if you execute 100 units in period 0, and 200 units in period 1, you estimate twice as much setup time in period 1 as period 0. This is particularly applicable in cases where a fixed lot size is being applied, either by WIT itself or by some execution system (even a manual one). Typically manufacturing environments with significant setup times are managed using lot sizing.

WIT Model

This model is compatible with all WIT solution functions.

For each BOM entry and substitute BOM entry whose usage causes a setup, add an amount to the usageRate such that

execVol * usageRate ≈ processing time + setup time

For example, if the execution system uses a batch size of 10 and the setup time is 15, add 1.5 (= 15/10) to the usageRate.

Once-Per-Family

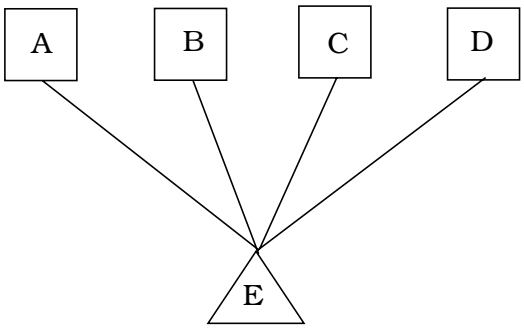
Problem Definition/Assumptions

The period length is short enough that the setup time incurred by a BOM entry is incurred either once or not at all.

The BOM entries that consume a particular capacity are grouped into families and a setup time is associated with each family. The setup time for a family is incurred at most once per period and it is incurred if and only if at least one of the BOM entries is used during the period.

The following is an example of a once-per-family setup problem.

FIGURE 12 Example Setup Problem: Once-Per-Family



Families: Family “AE” includes just the BOM entry from A to E and has a setup time of 15. Family “CDE” includes the BOM entries from C to E and from D to E and has a setup time of 25. The BOM entry from B to E is in no family. Thus family AE consumes an additional 15 units of capacity E’s supply in any period in which operation A is executed and family CDE consumes an additional 25 units of capacity E’s supply in any period in which operation C or operation D (or both) are executed or both and this is

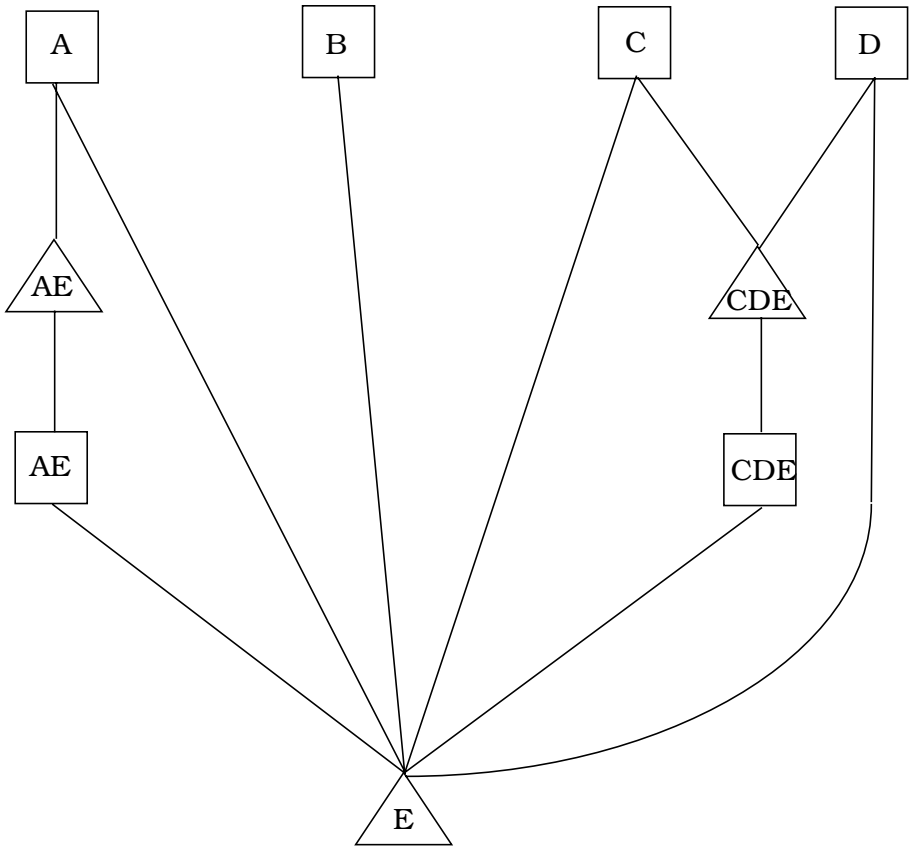
independent of the execution volume. Execution of operation B does not incur any setup time.

WIT Model

This model is compatible only with WIT solution functions that respect lot sizing, i.e., heuristic implosion, WIT-MRP, and FSS, but not optimizing implosion.

The model is formed by attaching one regulator for each family. It's easiest to explain the model by illustrating its application to the example problem.

FIGURE 13 Example of the Once-Per-Family Setup Model



Operations AE and CDE are called setup operations. Capacities AE and CDE are called setup capacities. The usageRate on the BOM entries connecting the original operations to the setup capacities (in this case A--AE, C--CDE, and D--CDE) is 1. There is no supply of the setup capacities. The setup operations have minLotSize and incLotSize = 1. The usageRates on the BOM entry connecting each setup operation to the original capacity is equal to the corresponding setup time. Thus the usageRate on BOM entry AE--E is 15 and the usageRate on BOM entry CDE--E is 25. The prodRate on the BOP entries connecting each setup operation with its setup capacity is some very large number, say 1000000.

Now consider a scenario of the use of this model with heuristic implosion. Suppose the first time capacity E is consumed in some period is by operation D, which has an execVol of 7. Suppose the usageRate on the BOM entry D--E is 2. Then BOM entry D--E causes of consumption of 2 * 7 = 14 units of supply of capacity E. BOM entry D--CDE causes a requirement for 7 units of supply of capacity CDE. Since capacity CDE has no supply, more volume of it must be produced. The only way to do this is to execute operation CDE. Operation CDE has a minLotSize of 1, which produces enough of capacity CDE, so 1 unit of operation CDE is executed. This produces 1000000 units of capacity CDE of which 7 are consumed leaving 999993 as net supply. BOM entry CDE--E causes the consumption of 25 units of supply of capacity E. Thus capacity E has had 14 units consumed for processing time and 25 units consumed for setup time.

Suppose the next time operation D is executed in the period, it is with an execVol of 12. The result: 24 units of capacity E are consumed for processing time and 12 units of the 999993 available of capacity CDE are consumed, but operation CDE is not executed again and so no further setup time is incurred. Because of this large newly available supply of capacity CDE, operation CDE is never again executed in that period, even if operation C is executed. Thus the setup time is incurred exactly once. Finally, the large volume of the setup capacity CDE is discarded at the end of the period, and so if operations C or D are executed in the next period, the setup operation CDE must be executed again.

Notes:

- After an implosion, to determine which setups where incurred when, simply look at the execVols of the setup operations. ex-

ecVol = 1, means the setup was incurred in the period; 0 means it wasn't; anything else is an error.

- The offsets on the original BOM entries are also applied to the BOM entries connecting the original operations to the setup capacities (D-CDE, etc.).
- This model can be extended to the case of setups on subsitute BOM entries.

Assessment

The constant estimate and linear estimate models are probably sufficient for most WIT models. They assume a longer period length, which is consistent with the way WIT is designed to be used. The once-per-family model assumes a short period length, which might lead other difficulties with WIT. And of course, the once-per-family model cannot be used with optimizing implosion and doesn't apply to substitutes.

Variable Length Periods

WIT's periods are considered to be of uniform length. For example, each period in a WIT model might represent a week, or each period might represent a day. However, often a user will want to apply WIT with variable length periods. Thus a user might want to build a WIT model in which each of the first 10 periods represents a day, each of the next 6 periods represents a week and each of the last 4 periods represents a month.

While WIT does not explicitly handle variable length periods, in many cases, it can solve problems that have variable length periods, if the input data is suitably preprocessed, as described below.

(Note that this technique is not structural in nature and does not use regulators.)

Limitations

If variable length periods are to be modeled in WIT, the following restrictions apply:

1. Optimizing implosion with objective #2 cannot be used.
2. If either of the following attributes is set to a value other than its default, the value may have a somewhat odd interpretation:
   - buildAheadLimit
   - shipLateLimit

   See "Special Case: Scalar Period Attributes" on page 26.

Preprocessing

We define the "raw" data (before preprocessing for variable length periods) in terms of some other time units, which will be called "raw periods". The raw periods are required to be of uniform duration. For this discussion, the term "period" will refer to the period that WIT will see (which will represent a variable amount of time). By the convention, the first raw period is raw period 0, similar to the convention for periods.

The relationship between raw periods and periods is defined by the following:

- nPeriods:

  The number of (variable length) periods (usual definition).

- periodLength[t]:
  The number of raw periods in period t.
  An integer ≥ 1
  The duration of a raw period should be made short enough that periodLength[t] is an integer.

Given the defining quantities above, the following derived quantities can be computed:

- lastRawPeriod[t]:

$$\mathbf{lastRawPeriod}[\mathbf{t}] = \left(\sum_{\mathbf{t}'=0}^{\mathbf{t}} \mathbf{periodLength}[\mathbf{t}'] \right) - 1$$

  lastRawPeriod[t] is the raw period that ends when period t ends.

- period[r]:

$$\mathbf{period}[\mathbf{r}] = \mathbf{min}\{\mathbf{t} | \mathbf{lastRawPeriod}[\mathbf{t}] \geq \mathbf{r}\}$$

  period[r] is the period that contains rawPeriod r.

- rawPeriods[t]:

$$\mathbf{rawPeriods}[\mathbf{t}] = \{\mathbf{r} | \mathbf{period}[\mathbf{r}] = \mathbf{t}\}$$

  rawPeriods[t] is the set of raw periods contained in period t.

Using the above expressions, we can define the various WIT attributes that have a period aspect to them from the corresponding attributes defined in terms of raw periods. The manner in which this calculation is done can be grouped into 4 categories: period mapping, raw period mapping, summation, and special cases.

Period Mapping

An example an attribute that should be computed by period mapping is earliestPeriod for a BOM entry:

```
earliestPeriod = period[rawEarliestPeriod]
```

where rawEarliestPeriod is earliestPeriod expressed as a raw period.

The following attributes should be computed by period mapping:

TABLE 1 Attributes To Be Computed By Period Mapping

Attribute	Object Type
earliestPeriod	BOM Entry
earliestPeriod	Substitute BOM Entry
earliestPeriod	BOP Entry
focusHorizon	Demand
latestPeriod	BOM Entry
latestPeriod	Substitute BOM Entry
latestPeriod	BOP Entry

Raw Period Mapping

An example an attribute that should be computed by raw period mapping is yieldRate for an operation:

```
yieldRate[t] = rawYieldRate[lastRawPeriod[t]]
```

where rawYieldRate[r] is the yieldRate in raw period r.

Alternatively, one could use the average of the rawYieldRates in the period, or the rawYieldRate for a different rawPeriod contained in the period, e.g., the median raw period for the period. This may also apply to some of the other attributes for which raw period mapping is appropriate, depending on the meaning of the attribute.

The following attributes should be computed by raw period mapping:

TABLE 2 Attributes To Be Computed By Raw Period Mapping

Attribute	Object Type
cumShipBounds	Demand
incLotSize	Operation
minLotSize	Operation
obj1ExecCost	Operation
obj1ScrapCost	Part
obj1ShipReward	Demand
obj1SubCost	Substitute BOM Entry
priority	Demand
stockBounds	Part
yieldRate	Operation

Summation

An example an attribute that should be computed by summation is supplyVol for a part:

$$\mathbf{supplyVol}[t] = \sum_{r \in \mathbf{rawPeriods}[t]} \mathbf{rawSupplyVol}[r]$$

where rawSupplyVol[r] is the supplyVol in raw period r. The following attributes should be computed by summation:

TABLE 3 Attributes To Be Computed By Summation

Attribute	Object Type
demandVol	Demand
execBounds	Operation
execVol	Operation
fssShipVol	Demand
obj1CumShipReward	Demand
obj1StockCost	Part
shipVol	Demand
subVol	Substitute BOM Entry
supplyVol	Part

Special Case: Scalar Period Attributes

The demand attributes buildAheadLimit and shipLateLimit are scalar attributes whose are values numbers of periods. Ideally, these should be set to values that vary with the period, to reflect varying period length, but since they are scalar valued, this can't be done. Note that the extreme values of 0 and nPeriods - 1 are meaningful in the variable length period case and that by default, the attributes take on extreme values. However, if an intermediate value is used, that value will represent a different amount of real time in different periods. This may or may not be desirable.

These attributes might be computed by arbitrarily choosing a period, t, and scaling on that basis:

$$\mathbf{shipLateLimit} = \frac{\mathbf{rawShipLateLimit}}{\mathbf{periodLength}[\mathbf{t}]}$$

where rawShipLateLimit is the shipLateLimit expressed in units of raw periods. Since the first periods in a WIT model are normally of most interest to the user, setting t = 0 might be appropriate.

Special Case: offset

The most complicated attribute to compute in the variable length period context is offset (for BOM entries, substitute BOM entries, and BOP entries). The "raw" data for offset is given by:

- rawRawOffset[r]:

  The offset in raw period r, expressed in units of raw periods. This should be an integer. It can be positive, negative for zero.

The calculation of offset can be described by defining a number of intermediate quantities

- rawOffset[t]:

  The offset in period t, expressed in units of raw periods. This is calculated from rawRawOffset by raw period mapping:

$$\mathbf{rawOffset}[\mathbf{t}] = \mathbf{rawRawOffset}[\mathbf{lastRawPeriod}[\mathbf{t}]]$$

- resultRawPeriod[t]:

  The result of applying rawOffset:

$$\mathbf{resultRawPeriod}[\mathbf{t}] = \mathbf{lastRawPeriod}[\mathbf{t}] - \mathbf{rawOffset}[\mathbf{t}]$$

- resultPeriod[t]:
  resultRawPeriod[t] mapped back into a period.

$$\mathbf{resultPeriod}[t] = \mathbf{period}[\mathbf{resultRawPeriod}[t]]$$

Finally, offset[t] is computed as follows:

$$\mathbf{offset}[t] = t - \mathbf{resultPeriod}[t] + \frac{\mathbf{lastRawPeriod}[\mathbf{resultPeriod}[t]] - \mathbf{resultRawPeriod}[t]}{\mathbf{periodLength}[\mathbf{resultPeriod}[t]]}$$

The first two terms in the above expression give the integer part of offset[t] as the offset necessary to move from t to resultPeriod[t]. The last term gives the fractional part of offset[t] as the offset necessary to move from the end of resultPeriod[t] to the point in time corresponding to resultRawPeriod[t].

Example offset Computation

The following example may serve to illustrate the computation of the offset attribute:

Given:

$$\mathbf{nPeriods} = 5$$
$$\mathbf{periodLength}[0] = 5$$
$$\mathbf{periodLength}[1] = 5$$
$$\mathbf{periodLength}[2] = 5$$
$$\mathbf{periodLength}[3] = 10$$
$$\mathbf{periodLength}[4] = 10$$
$$\mathbf{rawRawOffset}[34] = 27$$

We wish to compute offset[4]:

$$\textbf{lastRawPeriod}[4] = 34$$
$$\textbf{rawOffset}[4] = 27$$
$$\textbf{resultRawPeriod}[4] = 34 - 27 = 7$$
$$\textbf{period}[7] = 1$$
$$\textbf{resultPeriod}[4] = 1$$
$$\textbf{lastRawPeriod}[1] = 9$$
$$\textbf{offset}[4] = 4 - 1 + \frac{9 - 7}{5} = 3.4$$