

Rule Modeling Challenge November 2017 Soldier Pay

Corticon - Mike Parish

During different service periods a soldier may have the following characteristics:

Rank {private, corporal, sergeant, lieutenant, captain}

Profession {fighter, driver, cook, officer}

Service Type {active, reserve, retired}

Unit {HQ, paratroopers, marines, infantry}

Combat {yes, no}

Pay rate is determined by aggregating the amounts according to these rules:

Base rate is \$1/hr.

Private \$1/hr., corporal \$2/hr., sergeant \$3/hr., lieutenant \$4/hr., captain \$5/hr.

Fighter \$2/hr., driver \$1/hr., cook \$1/hr., officer \$3/hr.

Active \$2/hr., Reserve \$1/hr.

HQ \$1/hr., others \$2/hr.

Combat \$5/hr., non-combat \$0/hr.

Example:

Rank: Private 1/1/2015-12/31/2015,

Profession: Fighter 1/1/2015-6/30/2015, Cook 7/1/2015-12/1/2015

Service Type: Active 1/1/2015-6/30/2015, Reserve 7/1/2015-12/1/2015

Unit: HQ 1/1/2015-12/31/2015

Combat: 4/1/2015-6/30/2015

On 6/1/2015 he was a private, a fighter, on active duty, at HQ, in combat, So, his pay rate was $1(\text{base})+1(\text{private})+2(\text{fighter})+2(\text{active})+1(\text{HQ})+5(\text{combat}) = \$12/\text{hr}$

The challenge: assemble a single timeline for the soldier over a given service period that shows his/her hourly pay rate in any given time. Flag any conflicting dates (e.g. can't be a sergeant and a lieutenant at the same time).

Additional challenge: What are all the different aggregated pay rates that apply and during which periods?

An Example

The diagram shows a number of periods in a soldier's life.

In each period you can see the rank, profession, service type, unit and combat status along with the pay rates that applied at the time.

Base	\$1		
Rank	private (\$1)		
Profession	fighter (\$2)	cook (\$1)	
Service	active (\$2)	reserve (\$1)	
Unit	HQ (\$1)		
Combat	no (\$0)	yes (\$5)	no (\$0)
Aggregate	\$7	\$12	\$5

The soldier's total pay rate in any period is the sum of the components

Notes:

1. Reserve status starts immediately after active status to avoid the gap in the original problem statement
2. The unstated non-combat periods have been added.

Let's start with the simplest case.

Assume the pay rates and the soldier characteristics **do not change** within a given period.

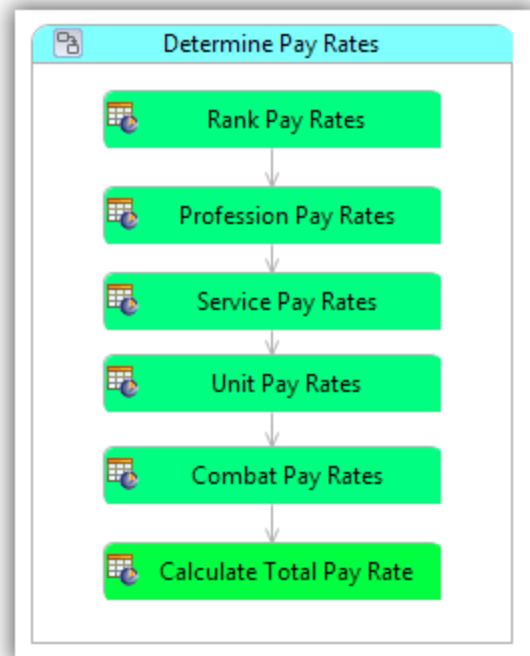
The problem is then very simple.

Total pay rate = base pay + rank pay + profession pay + service pay + unit pay + combat pay

And all we need to do is define the rules that specify the various pay amounts.

We can do this in several ways

1. Separate decision tables for each characteristic
2. Lookup tables defined in a rule sheet
3. Lookup tables defined in a database



In this example we'll use method #1

For example, the rules for rank pay would be something like this:

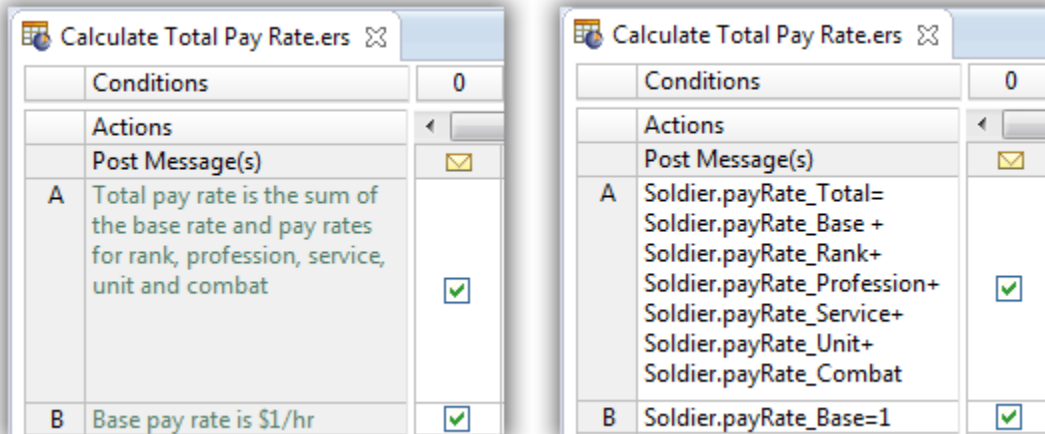
*Rank Pay Rates.ers		1	2	3	4	5
Conditions						
a	What is the soldier's rank?	'private'	'corporal'	'sergeant'	'lieutenant'	'captain'
b						
Actions						
Post Message(s)						
A	Rank pay rate is	1	2	3	4	5

This can be implemented as:

*Rank Pay Rates.ers		1	2	3	4	5
Conditions						
a	Soldier.rank	'private'	'corporal'	'sergeant'	'lieutenant'	'captain'
b						
Actions						
Post Message(s)						
A	Soldier.payRate_Rank	1	2	3	4	5

And there would be similar rule sheets for each of the other characteristics (see appendix).

A final rule sheet like this would then set the base rate and calculate the total pay rate



This is not very challenging however. All of the rule engines on the market could solve this just as easily.

What makes this problem more interesting is the fact that the soldier's characteristics are not static. They change over time. At different times they may have different combinations of rank, profession, service, unit and combat. Even the pay rates also could change over time, but for now we'll keep them static.

So a soldier's history will consist of many records each of which specifies a time period (start and end), the characteristic (rank, profession, service, unit, combat) and the value of that characteristic (such as private, corporal etc.)

Let's start with a simple case using just rank and profession:

1/1/2015-12/31/2015 Rank = private
1/1/2015-6/30/2015 Profession = fighter
6/30/2015-12/31/2015 Profession = cook

Although three records are required to capture the soldier's history there are really only TWO time periods involved:

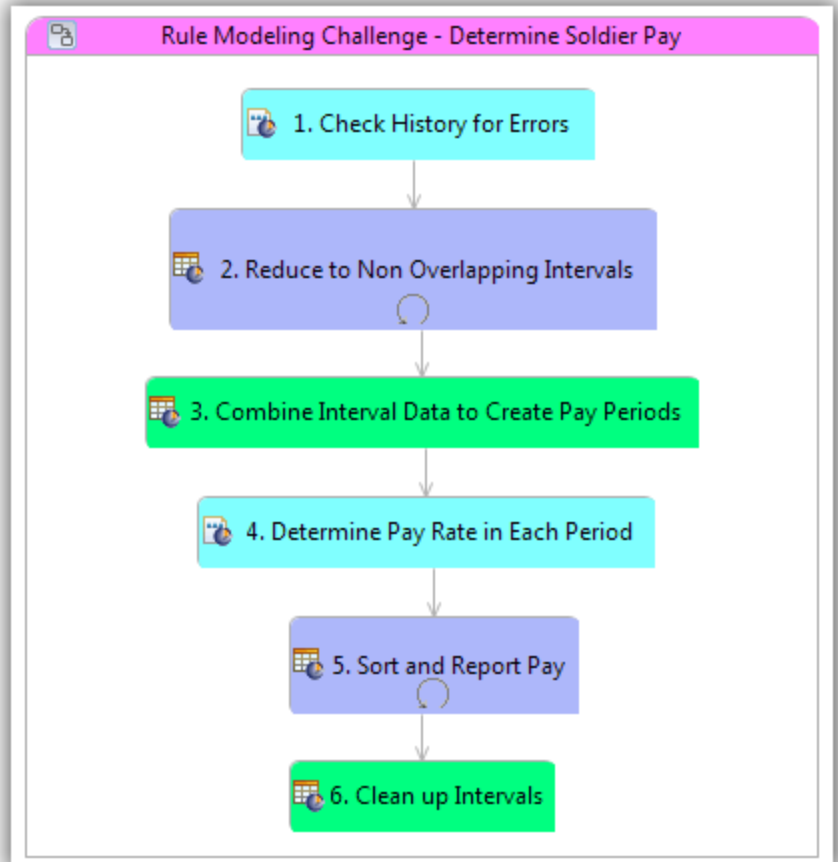
1/1/2015-6/30/2015 during which he was a private and a fighter pay=1+1+2
6/30/2015-12/31/2015 during which he was a private and a cook pay=1+1+1

So what we need to do is find a way to condense the separate time periods which contain a single characteristic into a combined time period which contains all five of the characteristics for that time period. Then we can simply send that set of records to the previously described "Determine Pay Rates" to get the total pay in each period.

So this part of the problem boils down to taking a set of overlapping intervals and finding the non-overlapping set that covers the entire period. Then we can go through and collect up the five characteristics in each period.

The entire process might look like this.

1. Check the data to ensure the dates are logical and that values are correct and that there are no gaps or overlaps within a characteristic.
2. Reduce the history to non-overlapping intervals. This is done by taking all pairs of overlapping intervals and splitting as necessary to remove the overlaps. This continues until there are no overlaps.
3. Merge the data from the five intervals (rank, profession, service, unit and combat) that have the same start and end.
4. Calculate the total pay for each of those intervals
5. Sort into ascending order of start date for reporting purposes
6. Clean up the original history records that have now been merged into the pay periods.



We'll just examine the details of #2 and #3. In the appendix will be some discussion of how we write rules to identify gaps and overlaps in intervals. This is something that has widespread applicability in applications that handle dates (which is probably most of them)

Reduce History to Non-Overlapping Intervals

Using the simple example from before

1/1/2015-12/31/2015 Rank = private
1/1/2015-6/30/2015 Profession = fighter
6/30/2015-12/31/2015 Profession = cook

We can see that the first record (Rank=private) spans two Professions.

So we first split the rank record into two parts to match the fighter and cook periods.

This results in four records with no overlaps

1/1/2015-6/30/2015 Rank = private
6/30/2015-12/31/2015 Rank = private
1/1/2015-6/30/2015 Profession = fighter
6/30/2015-12/31/2015 Profession = cook

Now we can match up the starts and ends and merge the characteristics:

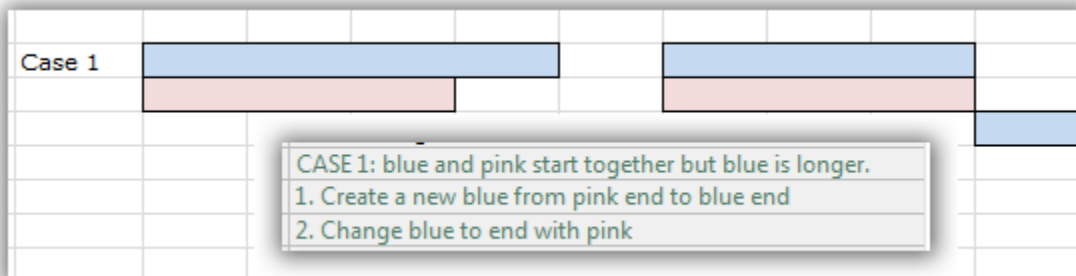
1/1/2015-6/30/2015 during which he was a private and a fighter
6/30/2015-12/31/2015 during which he was a private and a cook

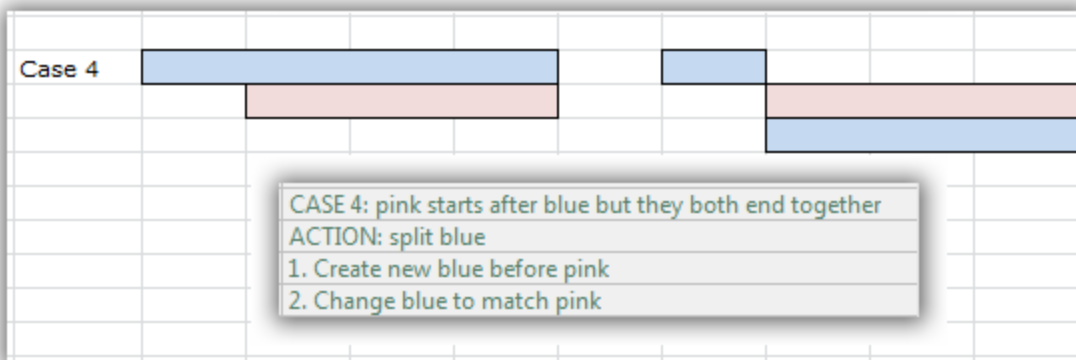
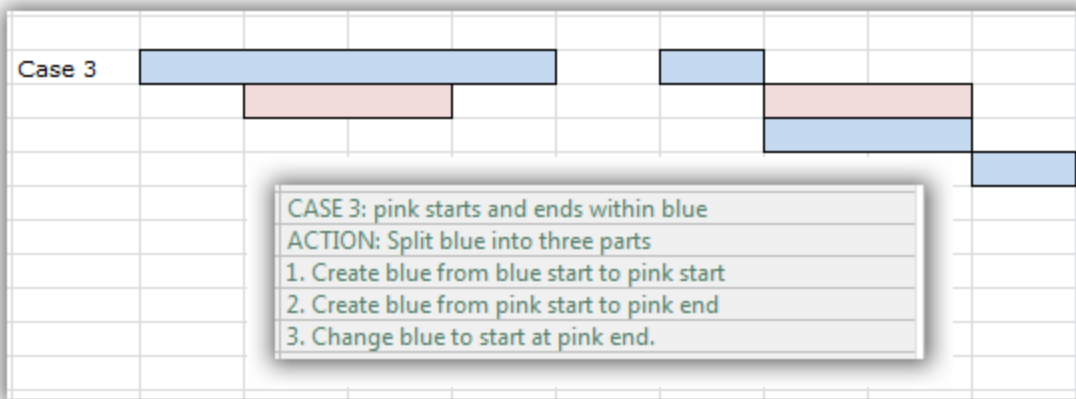
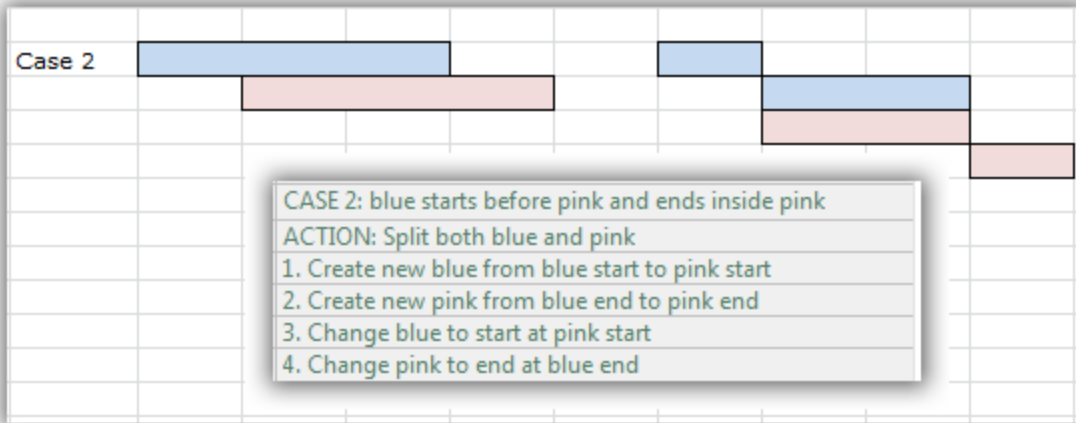
Of course in the full problem we are doing this with five characteristics and possibly many more overlaps, so how would Corticon deal with this?

First we observe that there are only six ways in which two intervals (blue and pink) can overlap:

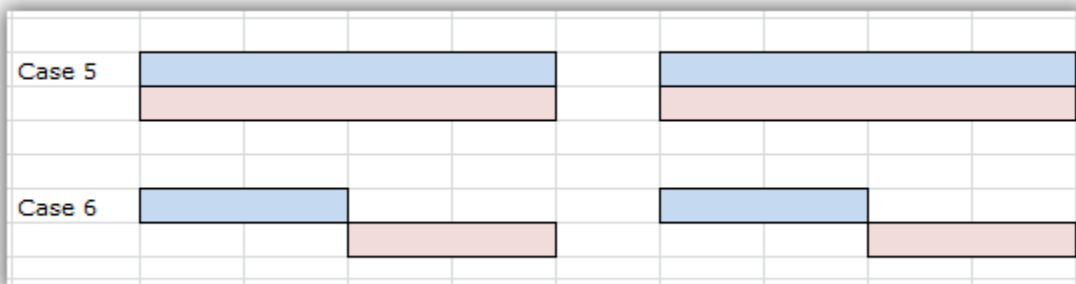
CASE 1: both start together but blue is longer;	Split the blue at the end of pink.
CASE 2: pink starts inside blue and ends after;	Split both at the overlap;
CASE 3: pink is entirely within blue;	Split blue into 3 parts.
CASE 4: pink starts after blue but they end together;	Split blue.
CASE 5: exact overlap;	No action required
CASE 6: no overlap;	No action required.

This may be easier to follow with a diagram





The other two cases don't require any action:



The rules need to locate the cases on the left and convert them to those on the right.

In decision table format it might look like this (in Natural Language format)

Reduce to Non Overlapping Intervals.ers		1	2	3	4
Conditions					
a	Blue and Pink start together	T	-	-	-
b	Blue ends after Pink ends	T	-	T	-
c	Blue starts before Pink starts	-	T	T	T
d	Blue ends before Pink ends	-	T	-	-
e	Blue ends after Pink starts	-	T	-	-
f	Blue and Pink end together	-	-	-	T
g	Blue ends before Pink starts	-	-	-	-
h					
Actions					
Post Message(s)		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A	CASE 1: blue and pink start together but blue is longer.				
B	ACTION: split blue at the end of pink				
C	1. Create a new blue from pink end to blue end	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
D	2. Change blue to end with pink	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
E					
F	CASE 2: blue starts before pink and ends inside pink				
G	ACTION: Split both blue and pink				
H	1. Create new blue from blue start to pink start	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I	2. Create new pink from blue end to pink end	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
J	3. Change blue to start at pink start	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
K	4. Change pink to end at blue end	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
L					
M	CASE 3: pink starts and ends within blue				
N	ACTION: Split blue into three parts				
O	1. Create blue from blue start to pink start	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
P	2. Create blue from pink start to pink end	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Q	3. Change blue to start at pink end.	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
R					
S	CASE 4: pink starts after blue but they both end together				
T	ACTION: split blue				
U	1. Create new blue before pink	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
V	2. Change blue to match pink	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

One possible implementation is:

Conditions		1	2	3	4
a	blue.start=pink.start	T	-	-	-
b	blue.end > pink.end	T	-	T	-
c	blue.start < pink.start	-	T	T	T
d	blue.end < pink.end	-	T	-	-
e	blue.end > pink.start	-	T	-	-
f	blue.end = pink.end	-	-	-	T
g	blue.end <= pink.start	-	-	-	-
h					

Actions		1	2	3	4
Post Message(s)		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A	Interval.new[type=blue.type, name=blue.name,start=pink.end,end=blue.end]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
B	blue.end=pink.end	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
C		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
D	Interval.new[type=blue.type, name=blue.name,start=blue.start,end=pink.start]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
E	Interval.new[type=pink.type, name=pink.name,start=blue.end,end=pink.end]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
F	blue.start=pink.start	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
G	pink.end=blue.end	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
H		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I	Interval.new[type=blue.type, name=blue.name,start=blue.start,end=pink.start]	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
J	Interval.new[type=blue.type, name=blue.name,start=pink.start,end=pink.end]	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
K	blue.start=pink.end	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
L		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
M	Interval.new[type=blue.type, name=blue.name,start=blue.start,end=pink.start]	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
N	blue.start = pink.start	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

But

why does this rule sheet only refer to two intervals (blue and pink) when there are five characteristics (each with its own set of intervals) to be handled in the challenge.

Why don't we need to handle all the possible overlaps between five intervals (a large number)?

It's because Corticon will automatically apply this rule sheet if ANY two intervals overlap (including any created during the execution of these rules) and it will continue doing so (two at a time) until there are no more overlaps. It would work just as well even if there were one hundred different characteristics (though it might take a little longer).

So basically we just need to model the rules for handling two intervals and then we let Corticon worry about what to do when there are more.

Combine Intervals to Create Pay Periods

Now that we have eliminated any overlaps, we can gather up the five records (Rank, Profession, Service, Unit, and Combat) that have the same start and end and combine them into one.

Here's the Natural Language version

The screenshot shows the 'Combine Interval Data.ers' rule editor. The 'Scope' pane on the left lists several intervals: Interval [aCombat], Interval [aProfession], Interval [aRank], Interval [aService], Interval [aUnit], and Soldier. The 'Filters' pane contains five filters: 'Find all the Rank intervals', 'Find all the Profession intervals', 'Find all the Service intervals', 'Find all the Unit intervals', and 'Find all the Combat intervals'. The main editor area is divided into 'Conditions' and 'Actions' sections. The 'Conditions' section contains nine conditions (a-i) that find intervals matching the Rank start and end. The 'Actions' section contains one action (A) that creates a new Soldier pay period with all five types of data for the given interval.

Conditions	1
a Find the Profession interval that matches the Rank start	T
b Find the Service interval that matches the Rank start	T
c Find the Unit interval that matches the Rank start	T
d Find the Combat interval that matches the Rank start	T
e	
f Find the Profession interval that matches the Rank end	T
g Find the Service interval that matches the Rank send	T
h Find the Unit interval that matches the Rank end	T
i Find the Combat interval that matches the Rank end	T
j	

Actions	
Post Message(s)	
A Create a new Soldier pay period with all five types of data for the given interval	<input checked="" type="checkbox"/>

And the implementation:

The screenshot shows the 'Combine Interval Data.ers' rule editor with the implementation. The 'Scope' pane is the same as in the previous screenshot. The 'Filters' pane now contains five filters: 'aRank.type='Rank'', 'aProfession.type='Profession'', 'aService.type='Service'', 'aUnit.type='Unit'', and 'aCombat.type='Combat''. The main editor area is divided into 'Conditions' and 'Actions' sections. The 'Conditions' section contains nine conditions (a-i) that check if the Rank start and end match the start and end of the other intervals. The 'Actions' section contains one action (A) that creates a new Soldier pay period with all five types of data for the given interval.

Conditions	1
a aRank.start = aProfession.start	T
b aRank.start = aService.start	T
c aRank.start = aUnit.start	T
d aRank.start = aCombat.start	T
e	
f aRank.end = aProfession.end	T
g aRank.end = aService.end	T
h aRank.end = aUnit.end	T
i aRank.end = aCombat.end	T
.	

Actions	
Post Message(s)	
A Soldier.newUnique[start=aRank.start,end=aRank.end,rank=aRank.name,profession=aProfession.name,serviceType=aService.name,unit=aUnit.name,combatStatus=aCombat.name]	<input checked="" type="checkbox"/>

Note that during the execution of the rule the intervals in lines f-i are the same ones matched in a-d

A Test Case

Input

Nine input records are supplied

Input	
▶	Interval [1] {1/1/2015-12/31/2015 Rank=private}
▶	Interval [2] {1/1/2015-6/30/2015 Profession=fighter}
▶	Interval [3] {6/30/2015=12/31/2015 Profession=cook}
▶	Interval [4] {1/1/2015=6/30/2015 Service=active}
▶	Interval [5] {6/30/2015=12/31/2015 Service=reserve}
▶	Interval [6] {1/1/2015-12/31/2015 Unit=HQ}
▶	Interval [7] {1/1/2015-4/1/2015 Combat=no}
▶	Interval [8] {4/1/2015-6/30/2015 Combat=yes}
▶	Interval [9] {6/30/2015-12/31/2015 Combat=no}

Each of which has attributes like this

Interval [1] {1/1/2015-12/31/2015 Rank=private}
end [12/31/2015 12:00:00 AM]
name [private]
start [1/1/2015 12:00:00 AM]
type [Rank]

Output

Three output records are created

Soldier [3]
combatStatus [no]
end [4/1/2015 12:00:00 AM]
payRate_Base [1]
payRate_Combat [0]
payRate_Profession [2]
payRate_Rank [1]
payRate_Service [2]
payRate_Total [7]
payRate_Unit [1]
profession [fighter]
rank [private]
serviceType [active]
start [1/1/2015 12:00:00 AM]
unit [HQ]

Soldier [1]
combatStatus [yes]
end [6/30/2015 12:00:00 AM]
payRate_Base [1]
payRate_Combat [5]
payRate_Profession [2]
payRate_Rank [1]
payRate_Service [2]
payRate_Total [12]
payRate_Unit [1]
profession [fighter]
rank [private]
serviceType [active]
start [4/1/2015 12:00:00 AM]
unit [HQ]

Soldier [2]
combatStatus [no]
end [12/31/2015 12:00:00 AM]
payRate_Base [1]
payRate_Combat [0]
payRate_Profession [1]
payRate_Rank [1]
payRate_Service [1]
payRate_Total [5]
payRate_Unit [1]
profession [cook]
rank [private]
serviceType [reserve]
start [6/30/2015 12:00:00 AM]
unit [HQ]

Report

1/1/2015 12:00:00 AM-4/1/2015 12:00:00 AM Total (7)= Base (1) + private (1) + fighter (2) + active (2) + HQ (1) + no (0)
4/1/2015 12:00:00 AM-6/30/2015 12:00:00 AM Total (12)= Base (1) + private (1) + fighter (2) + active (2) + HQ (1) + yes (5)
6/30/2015 12:00:00 AM-12/31/2015 12:00:00 AM Total (5)= Base (1) + private (1) + cook (1) + reserve (1) + HQ (1) + no (0)

A More Complex Example

Input	
▷	Interval [1] {1/1/2015-3/1/2015 Rank=private}
▷	Interval [2] {1/1/2015-6/30/2015 Profession=fighter}
▷	Interval [3] {6/30/2015=12/31/2015 Profession=cook}
▷	Interval [4] {1/1/2015=6/30/2015 Service=active}
▷	Interval [5] {6/30/2015=12/31/2015 Service=reserve}
▷	Interval [6] {1/1/2015-10/31/2015 Unit=HQ}
▷	Interval [7] {1/1/2015-4/1/2015 Combat=no}
▷	Interval [8] {4/1/2015-6/30/2015 Combat=yes}
▷	Interval [9] {6/30/2015-12/31/2015 Combat=no}
▷	Interval [10] {10/31/2015-12/31/2015 Unit=paratroopers}
▷	Interval [11] {3/1/2015-7/1/2015 Rank=corporal}
▷	Interval [12] {7/1/2015-12/31/2015 Rank=sergeant}

Produces this result

1/1/2015 12:00:00 AM-3/1/2015 12:00:00 AM	Total (7)= Base (1) + private (1) + fighter (2) + active (2) + HQ (1) + no (0)
3/1/2015 12:00:00 AM-4/1/2015 12:00:00 AM	Total (8)= Base (1) + corporal (2) + fighter (2) + active (2) + HQ (1) + no (0)
4/1/2015 12:00:00 AM-6/30/2015 12:00:00 AM	Total (13)= Base (1) + corporal (2) + fighter (2) + active (2) + HQ (1) + yes (5)
6/30/2015 12:00:00 AM-7/1/2015 12:00:00 AM	Total (6)= Base (1) + corporal (2) + cook (1) + reserve (1) + HQ (1) + no (0)
7/1/2015 12:00:00 AM-10/31/2015 12:00:00 AM	Total (7)= Base (1) + sergeant (3) + cook (1) + reserve (1) + HQ (1) + no (0)
10/31/2015 12:00:00 AM-12/31/2015 12:00:00 AM	Total (8)= Base (1) + sergeant (3) + cook (1) + reserve (1) + paratroopers (2) + no (0)

Base	\$1					
Rank	private (\$1)	corporal (\$2)			sergeant (\$3)	
Profession	fighter (\$2)			cook (\$1)		
Service	active (\$2)			reserve (\$1)		
Unit	HQ (\$1)				para (\$2)	
Combat	no (\$0)		yes (\$5)		no (\$0)	
Aggregate	\$7	\$8	\$13	\$6	\$7	\$8

Notes:

1. Although the examples all start and end at midnight the same rules will work just fine for ANY time period, even as short as a few seconds or longer than a year.

Appendix

Pay Rule Sheets

*Rank Pay Rates.ers						
Conditions		1	2	3	4	5
a	Soldier.rank	'private'	'corporal'	'sergeant'	'lieutenant'	'captain'
b						
Actions						
Post Message(s)						
A	Soldier.payRate_Rank	1	2	3	4	5

Profession Pay Rates.ers					
Conditions		1	2	3	4
a	Soldier.profession	'fighter'	'driver'	'cook'	'officer'
b					
Actions					
Post Message(s)					
A	Soldier.payRate_Profession	2	1	1	3

Service Pay Rates.ers			
Conditions		1	2
a	Soldier.serviceType	'active'	'reserve'
b			
Actions			
Post Message(s)			
A	Soldier.payRate_Service	2	1

Unit Pay Rates.ers			
Conditions		1	2
a	Soldier.unit	'HQ'	other
b			
Actions			
Post Message(s)			
A	Soldier.payRate_Unit	1	2

Combat Pay Rates.ers			
Conditions		1	2
a	Soldier.combatStatus	'yes'	'no'
b			
Actions			
Post Message(s)			
A	Soldier.payRate_Combat	5	0

If there were more values within a pay rate we'd just add more columns to the appropriate rule sheet.

If there were more components to a soldier's pay then we'd just add more rule sheets.

We might enhance these rule sheets with an additional column for unrecognized values and default the pay rate to zero or issue an error message, though often it make the rule sheets less cluttered if the data validation and error reporting is handled in a separate rule sheet. Generally we'd use the validation rule sheets to flag the data as either "valid" or "invalid" and then set filters on the rule sheets to only process the valid data. This could be more efficient when there is a lot of data involved.

How to check for gaps in intervals

We need to verify that for each component (e.g. Rank) of the soldier's pay there are no gaps in the history.

Here's the Natural Language specification of the rules:

The screenshot shows the Corticon rule editor interface for a rule named '*IP_FindGaps in Soldier History.ers'. The interface is divided into several sections:

- Scope:** A tree view on the left showing a hierarchy: Gap, Interval [R1], Interval [R2], Interval [R3], Interval [X], and Interval [Y].
- Filters:** A table with 4 rows. Row 1: 'X ends before Y starts'. Row 2: 'X and Y are of the same type'. Rows 3 and 4 are empty.
- Conditions:** A table with 4 rows (a, b, c, d) and a status column. Row a: 'Case 1: For all intervals X and Y (X ends before Y starts) is there an interval that starts after X ends but before Y starts (overlaps X but not Y)'. Row b: 'Case 2: For all intervals X and Y (X ends before Y starts) is there an interval (other than X or Y) that ends after X ends but before Y starts (overlaps Y but not X)'. Row c: 'Case 3: For all intervals X and Y (X ends before Y starts) is there an interval (other than X or Y) that starts before X ends and ends after Y starts (Surrounds both X and Y)'. Row d is empty. All rows a, b, and c have a status of 'F'.
- Actions:** A table with 1 row (A) and a status column. Row A: 'Create a GAP record (from the end of X to the start of Y) and calculate its duration. Name it X-Y'. The status is checked.

Since Corticon is based on set theory principles it supports both the “exists” and “forAll” operators. These provide very powerful tools for writing rules about collections (sets) of objects.

Here is one possible implementation using the “exists” operator.

The screenshot shows the Corticon rule editor interface for the same rule '*IP_FindGaps in Soldier History.ers'. The implementation is more generic than the natural language version:

- Scope:** A tree view on the left showing a hierarchy: Gap, Interval [R1], Interval [R2], Interval [R3], Interval [X], and Interval [Y].
- Filters:** A table with 5 rows. Row 1: 'X.end < Y.start'. Row 2: 'X.type = Y.type'. Rows 3, 4, and 5 are empty.
- Conditions:** A table with 4 rows (a, b, c, d) and a status column. Row a: 'R1->exists(start>X.end, start<Y.start, R1.type=X.type)'. Row b: 'R2->exists(end>X.end, end<= Y.start, R2.type=X.type, name<>Y.name)'. Row c: 'R3->exists(start<=X.end, end>=Y.start, R3.type=X.type, name<>Y.name)'. Row d is empty. All rows a, b, and c have a status of 'F'.
- Actions:** A table with 1 row (A) and a status column. Row A: 'Gap.new[start=X.end,end=Y.start, type=X.type, name=X.name+'-' +Y.name, durationSeconds = X.end.secsBetween(Y.start)]'. The status is checked.

Also, this logic is entirely generic; it only refers to start, end, type and name and so can be used to check for gaps in any kind of interval. For example, insurance coverage periods, employment history etc.

Test Case

In this test case Corticon finds a 1 second gap between the rank of corporal and sergeant.

Input	Output
<ul style="list-style-type: none">Interval [1]<ul style="list-style-type: none">end [6/1/2017 12:00:00 AM]name [private]start [1/1/2017 12:00:00 AM]type [Rank]Interval [2]<ul style="list-style-type: none">end [12/1/2017 12:00:00 AM]name [sergeant]start [6/1/2017 12:00:02 AM]type [Rank]Interval [3]<ul style="list-style-type: none">end [6/1/2017 12:00:01 AM]name [corporal]start [5/31/2017 12:00:00 AM]type [Rank]Interval [4]<ul style="list-style-type: none">end [6/1/2017 12:00:01 AM]name [fighter]start [5/31/2017 12:00:00 AM]type [Profession]	<ul style="list-style-type: none">Interval [1]Interval [2]Interval [3]Interval [4]Gap [1]<ul style="list-style-type: none">durationSeconds [1]end [6/1/2017 12:00:02 AM]name [corporal-sergeant]start [6/1/2017 12:00:01 AM]type [Rank]

Message

- [1] PROBLEM: There's a gap in Rank between corporal and sergeant since all of the following are true
- [1] 1. there's nothing that starts between corporal and sergeant
- [1] 2. there's nothing that ends between corporal and sergeant
- [1] 3. there's nothing that surrounds the end of corporal and the start of sergeant
- [1] Gaps are only significant when they occur in the timeline of the same type of observation

How to check for overlaps in intervals

If two intervals overlap and are of the same type (e.g. Rank) but have different values (e.g. private and corporal) then that is considered an error.

Rules in Natural Language

Scope		Conditions	0	1	2
▶	Interval [blue]	a blue and pink are the same type		T	T
▶	Interval [pink]	b blue and pink have different names		T	T
		c pink starts with or after blue		T	T
		d pink starts before blue ends		T	T
		e pink ends before blue ends		T	F
		f			

Filters	Actions
1 blue <> pink	Post Message(s)

An implementation of the rules

Scope		Conditions	0	1	2
▶	Interval [blue]	a blue.type = pink.type		T	T
▶	Interval [pink]	b blue.name <> pink.name		T	T
		c pink.start >= blue.start		T	T
		d pink.start < blue.end		T	T
		e pink.end < blue.end		T	F
		f			

Filters	Actions
1 blue <> pink	Post Message(s)

The rule messages

Ref	Post	Alias	Text
1:2	Info	blue	CONFLICT: Interval {blue.start}-{blue.end} ({blue.type}={blue.name}) overlaps interval {pink.start}-{pink.end} ({pink.type}={pink.name})
1	Info	blue	CONFLICT: Between {pink.start} and {pink.end} {blue.type} was both {blue.name} and {pink.name}
2	Info	blue	CONFLICT: Between {pink.start} and {blue.end} {blue.type} was both {blue.name} and {pink.name}

This logic is also generic and could be used to check for overlaps in other kinds of interval.

Test Case

Input	Output
<ul style="list-style-type: none">Interval [1]<ul style="list-style-type: none">end [6/1/2017 12:00:00 AM]name [private]start [1/1/2017 12:00:00 AM]type [Rank]Interval [2]<ul style="list-style-type: none">end [12/1/2017 12:00:00 AM]name [corporal]start [6/1/2017 12:00:00 AM]type [Rank]Interval [3]<ul style="list-style-type: none">end [5/31/2017 12:00:00 AM]name [sergeant]start [5/30/2017 12:00:00 AM]type [Rank]	<ul style="list-style-type: none">Interval [1]<ul style="list-style-type: none">end [6/1/2017 12:00:00 AM]name [private]start [1/1/2017 12:00:00 AM]type [Rank]Interval [2]<ul style="list-style-type: none">end [12/1/2017 12:00:00 AM]name [corporal]start [6/1/2017 12:00:00 AM]type [Rank]Interval [3]<ul style="list-style-type: none">end [5/31/2017 12:00:00 AM]name [sergeant]start [5/30/2017 12:00:00 AM]type [Rank]

Rule Statements Rule Messages Properties Search Natural Language

Severity	Message
Info	[1] CONFLICT: Interval 1/1/2017 12:00:00 AM-6/1/2017 12:00:00 AM (Rank=private) overlaps interval 5/30/
Info	[1] Between 5/30/2017 12:00:00 AM and 5/31/2017 12:00:00 AM Rank was both private and sergeant

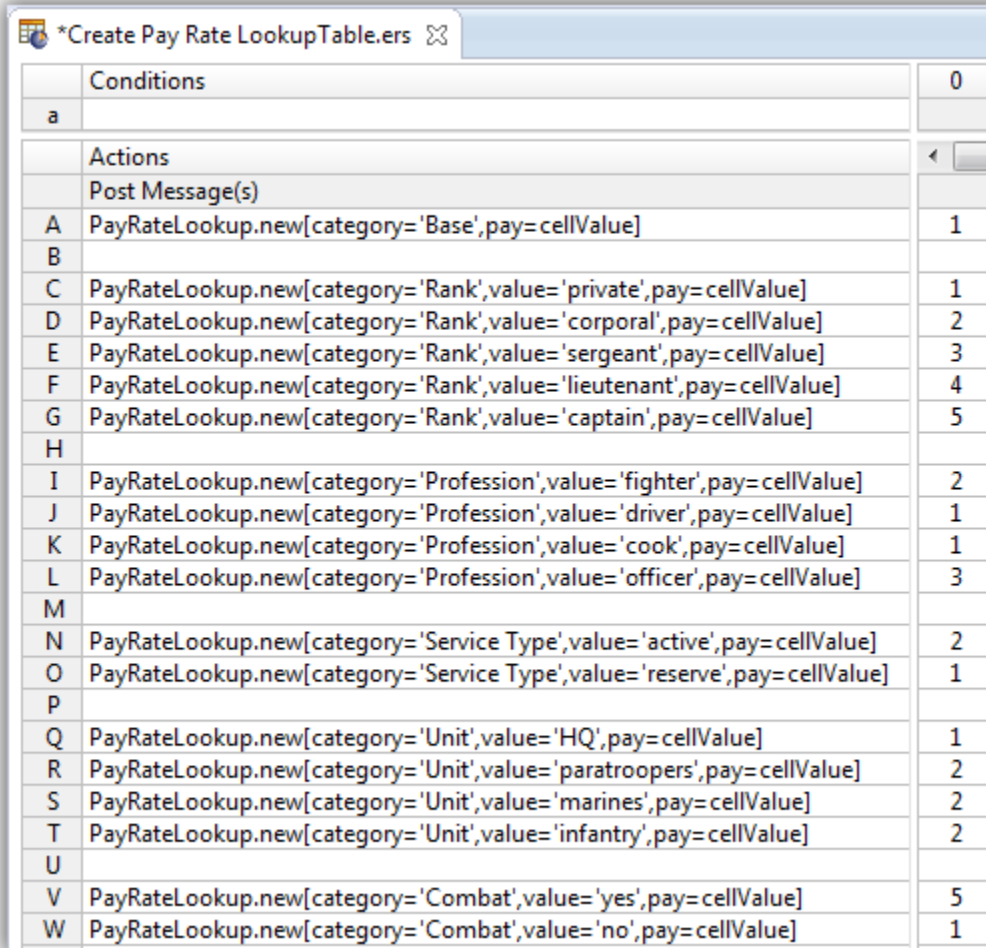
Alternative Method of Calculating Pay using Lookup Tables

An alternative approach is to use a lookup table for the pay rates.

This does not need a separate decision table for each of the different characteristics.

Instead a set of pay rate records is either read from a data base or created in memory using a rule sheet.

The in-memory creation might look like this:



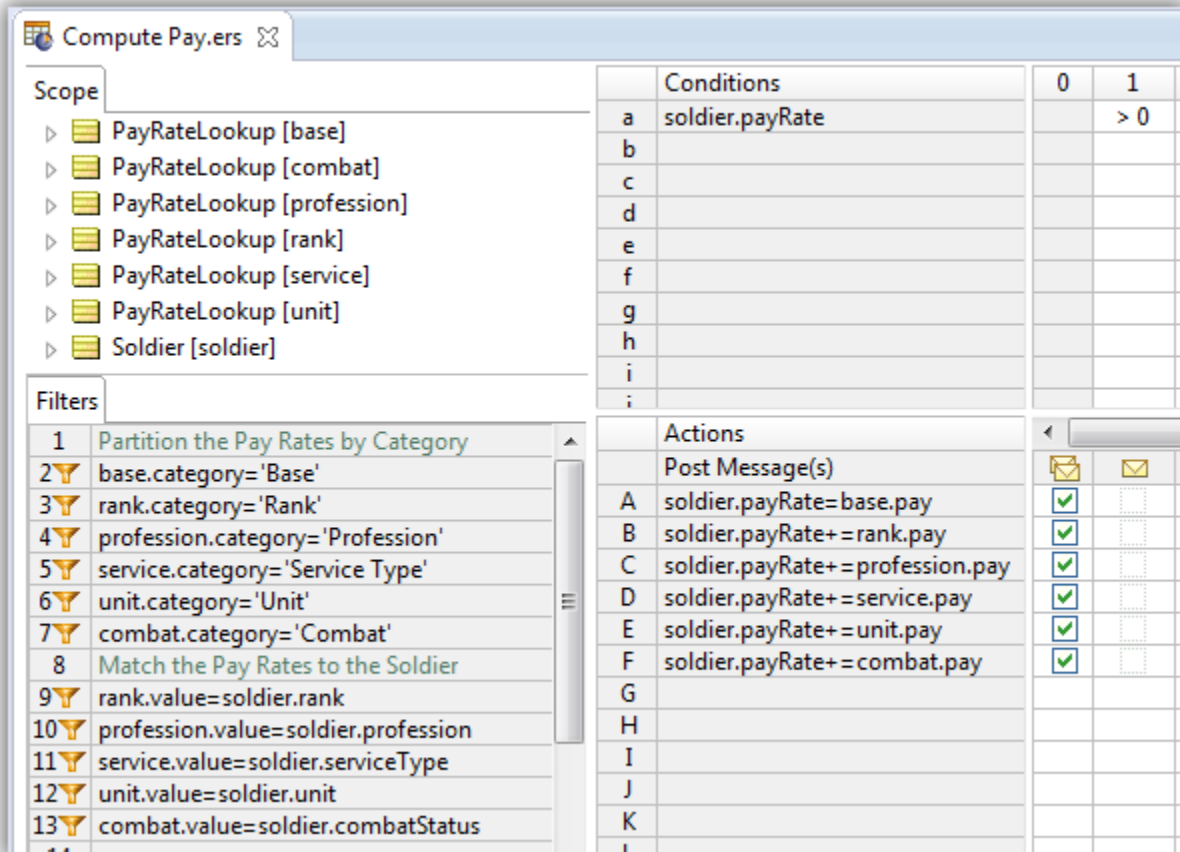
Conditions		0
a		
Actions		
Post Message(s)		
A	PayRateLookup.new[category='Base',pay=cellValue]	1
B		
C	PayRateLookup.new[category='Rank',value='private',pay=cellValue]	1
D	PayRateLookup.new[category='Rank',value='corporal',pay=cellValue]	2
E	PayRateLookup.new[category='Rank',value='sergeant',pay=cellValue]	3
F	PayRateLookup.new[category='Rank',value='lieutenant',pay=cellValue]	4
G	PayRateLookup.new[category='Rank',value='captain',pay=cellValue]	5
H		
I	PayRateLookup.new[category='Profession',value='fighter',pay=cellValue]	2
J	PayRateLookup.new[category='Profession',value='driver',pay=cellValue]	1
K	PayRateLookup.new[category='Profession',value='cook',pay=cellValue]	1
L	PayRateLookup.new[category='Profession',value='officer',pay=cellValue]	3
M		
N	PayRateLookup.new[category='Service Type',value='active',pay=cellValue]	2
O	PayRateLookup.new[category='Service Type',value='reserve',pay=cellValue]	1
P		
Q	PayRateLookup.new[category='Unit',value='HQ',pay=cellValue]	1
R	PayRateLookup.new[category='Unit',value='paratroopers',pay=cellValue]	2
S	PayRateLookup.new[category='Unit',value='marines',pay=cellValue]	2
T	PayRateLookup.new[category='Unit',value='infantry',pay=cellValue]	2
U		
V	PayRateLookup.new[category='Combat',value='yes',pay=cellValue]	5
W	PayRateLookup.new[category='Combat',value='no',pay=cellValue]	1

Doing it in a rule sheet makes it very easy to impose additional conditions on the pay rates.

We could instead read these values from a database. Corticon provides a means to read data directly from a wide variety of databases during the execution of a rule sheet without the need to write code or SQL.

If the pay rates varied over time then we'd include start and end dates too.

Then, instead of needing five rule sheets (one per characteristic), we can use a single generic rule sheet like this:



The filter section matches the appropriate lookup table record to the rank, profession, service, unit or combat and then the action section simply adds them up.

If the pay rates varied over time then the filter would have additional conditions that match not just on category and value but also on date.

How to Handle Variable Pay Rates

So far we have assumed that the pay rates are static. But it's quite possible that they too may have different intervals when they apply. In fact each different pay rate might have its own separate time line.

This is actually quite easy to incorporate into the current design.

Just as we reduced the soldier history to non-overlapping periods, so too we can reduce the pay rates to non-overlapping intervals.

Then we can combine both so that each interval contains the soldier characteristics and the pay rates that applied during that interval.