案例 5 机场码头

1 背景介绍

1．1 背景介绍

在机场，队列出现在很多不同的地点，比如说顾客售票区，行李选择区，餐馆，商店，顾客检票区等等。还有一个队列可能出现在飞机码头由于顾客的登机和下机而形成的。在繁忙或者延误的情况下，飞机可能在起飞前要等待很长一段时间。本案例是指向类似机场码头的研究。

1,2 情况介绍

这个码头有九个门，飞机可以在门前停靠以放下或者装载乘客以及他们的行李。每个门都可以处理所有的机型，但是每次只能停靠一架飞机。飞机被引导至门前并进行工作。这个工作包括让乘客下机，新乘客上机，加油等等。当所有完成后的一个很短时间内，飞机获得授权离开门。这个码头的系统可以这样说明：

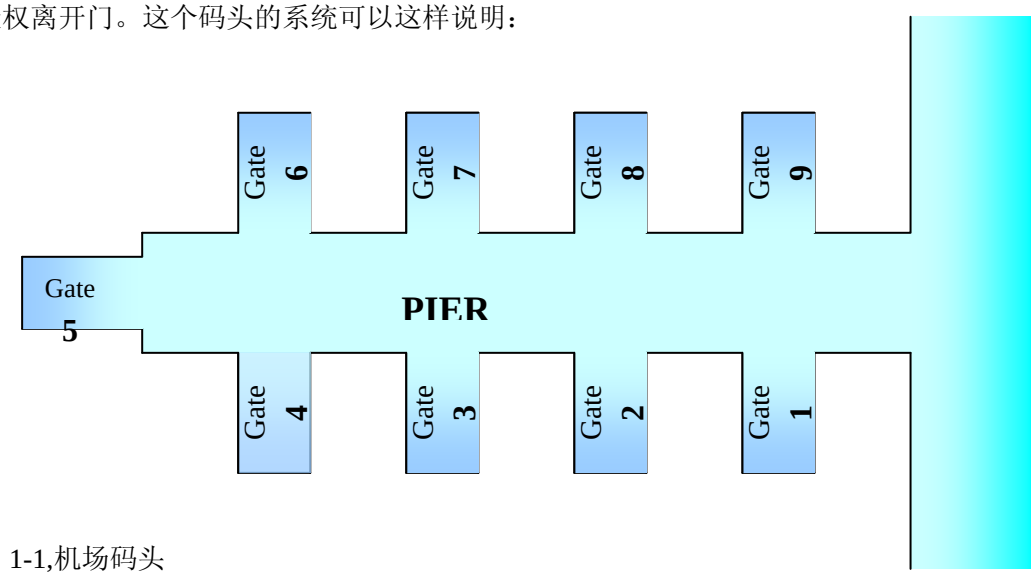


图 1-1,机场码头

这个机场码头每天从 6:00 开至第二天凌晨 2:00。最繁忙的时间段为 8:00 至 18:00，把这段时间设为调查时间段。在这个时间段内，每小时平均有 12 架飞机到达码头，服从负指数分布。一架飞机在门前停靠时间服从从 20 到 40 分钟的平均分布。

然而，在停靠期间，平均有四分之一的飞机会因为顾客或者行李登机而发生问题。若发生问题，飞机则会等待并占用该门直到顾客或者机场安全部门将问题解决，然后飞机才可以离开。这段额外的时间服从（30,40）分钟的平均分布。因此，这个时间必须被考虑。当有问题产生的时候，假设顾客和安全部门可以直接快速到达。

机场管理部门和航空公司对现状都感到不满，因为在这种情况下，相当数量的飞机需要等待授权降落和延迟。他们希望研究现状，通过确定平均有多少飞机在等待，以及平均等待时间是多久。（方案 1）

以下改变也需要研究：在卸载和装载后，发生问题的飞机被送至一个位于机场其他位置的中央处理区进行问题处理。管理人员希望飞机在码头的等待时间尽量的缩短。由于空间的限制，中央处理区域最多只能有四个机位。管理人员希望知道需要多少个位置。（方案 2）

管理人员假设专业化的管理可以更好的节约成本以及使码头有更好的流量。并提出以下几点：一支专门训练过的服务顾客队伍来移交问题。如果某架飞机发生问题，这个队伍就来处理。处理时间只需要（10,20）分钟的平均分布。管理人员想知道这在方案 3 和方案 4 中的效果。为了替代中央区域分派的方案，使用专门的团队的方案也是要被考虑的。

1,3 建议

1 建模前自己通过计算，尽可能的估计出门的利用率和排队等待时间的问题。
2 通过仿真手段，对管理人员针对现状提出的问题进行回答。（pier1.mod
3 通过仿真确定方案 2 是否合适。(pier2.mod
4 确定在两种情况下，只有一个顾客服务团队的效果。(pier3.mod

2 ED建模的建议

1,2,4 的建模都是基于 Source, Server, Queue, Sink 和 Multiservice 等基础原子的。3 中涉及到了顾客服务团队，需要用到 Operator 原子。当飞机出现问题的时候，该原子进行活动。飞机分为两种，分别是出问题的和没出问题的。有几种方式对此进行建模，分别为：

两个到达生成器，一个生成带问题的一个生成不带问题的。

一个到达生成器，使用伯努利分布定义问题飞机和无问题飞机。

在 Server 原子中才对飞机进行伯努利分布区分。

两种飞机到达码头的处理时间有不同的概率分布，这个可用不同的技术进行解决。一个比较简洁的例子是：

在 Sources 的 trigger on exit 中，输入:

`Label([cycletime], i) := probability distribution`,

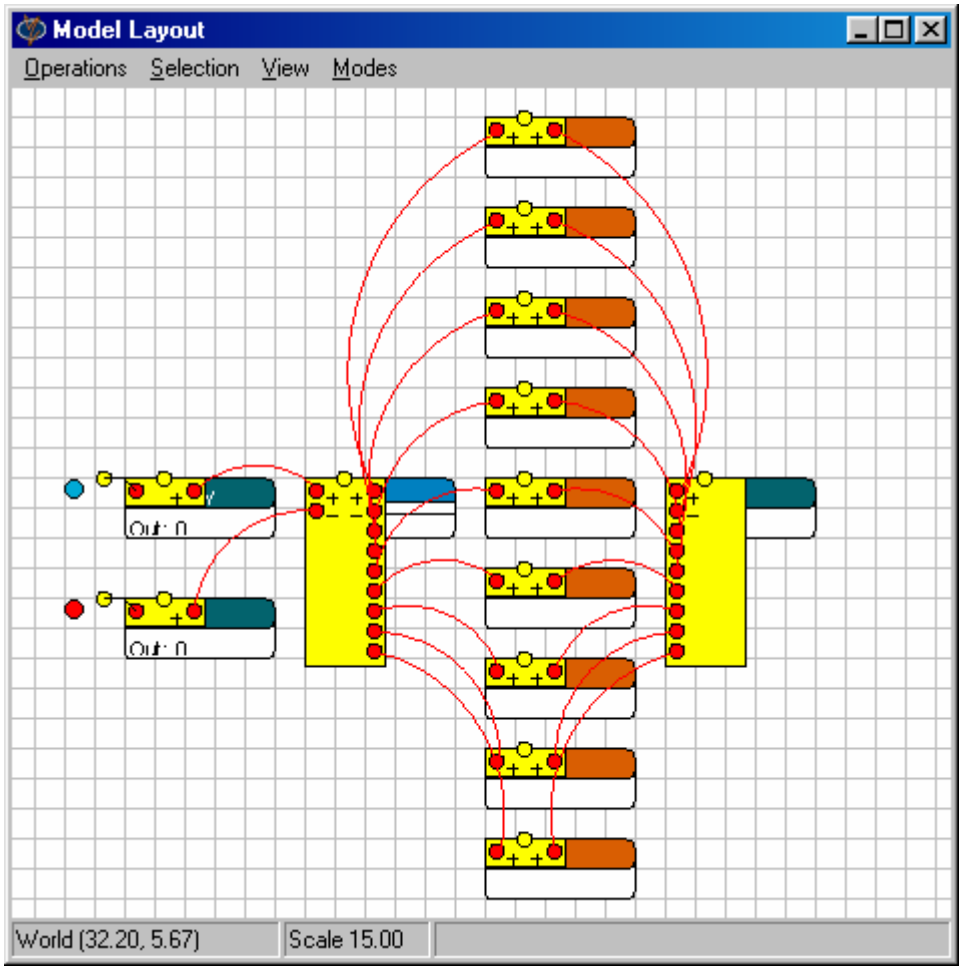
“probability distribution”用实际分布来代替。事实上，是通过给飞机定义一个 cycletime 的标签，然后 Servers 可以通过输入 label([cycletime],first(c))来读取标签上的值。

对于方案 3，operators 需要被召唤和释放在 Server 的触发器上。因此使用 Team 原子和 operator 原子结合着 4D 语句命令中的 calloperators 和 freeoperators 来实现。要注意这些 operators 只有在标准处理时间完成以后才会被召唤，因此将门拆分成两部分，一部分是正常的等待时间另一部分是额外的问题等待时间。这个通过使用两个前后相邻的 Servers 来分别代替第一部分和第二部分。使用 Closeinput 和 Closeoutput 来保证当第二部分忙碌的时候，没有飞机到达门。

为了计算门的平均利用率，可以使用“Experiment Wizard”功能。只需要添加一个包括所有门的组合并选择需要测量的数据就可以了。

3.模型的介绍

方案 1　　pier1.mod



到达生成部分，使用两个 Source 原子，无故障飞机生成原子叫做 No Delay，有故障飞机的生成原子叫做 Delay

No Delay：飞机没有任何问题。

Inter-arrival time: negexp(400) 平均每小时生成 9 架飞机

Trigger on exit: Label([cycletime], i) := uniform(mins(20),mins(40))当一架飞机离开此原子的时候，得到一个包含处理时间的标签。

Delay　　有问题的飞机

Inter-arrival time: negexp(1200)每小时生成 3 架飞机

Trigger on exit: Label([cycletime], i) := uniform(mins(20),mins(40)) + uniform(mins(30),mins(40))　当一架飞机离开此原子的时候，得到一个包含处理时间的标签。

Gate1 到 Gate9：码头的门

Cycletime: label([cycletime],first(c)) 在这里，读取标签 cycletime 的值

Experiment Wizard:实验

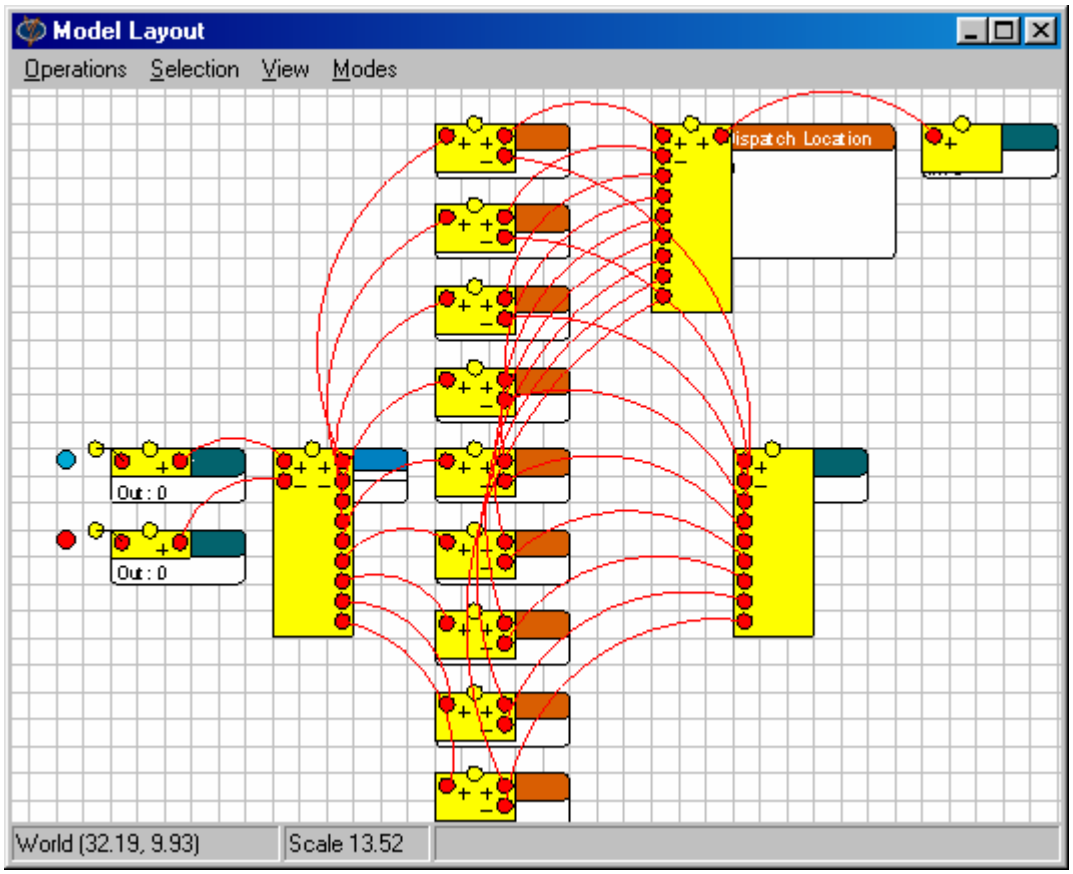
10 小时为一周期，2 小时的预热期，运行 100 个周期。

PFM1：门前的平均队长 avgcontent(cs),

PFM2：在门处的平均等待时间 avgstay(cs),

PFM3：门的平均利用率 定义一个包含所有门的组合，并选择运行测量数据进行测量。

方案 2 Pier2.mod



不需要再在 Sources 处设定 label 了，因为 9 个门的处理时间都是相似的。只有问题飞机才会得到一个 label。

Delay 有问题的飞机。

Trigger on exit: `Label([delay], i) := 1` 带有 1 的标签值的飞机是有问题的。

Gate1 到 Gate9 码头门

Cycletime: `uniform(mins(20),mins(40))`

Send to: `if(label([delay],first(c))=1,1,2)`

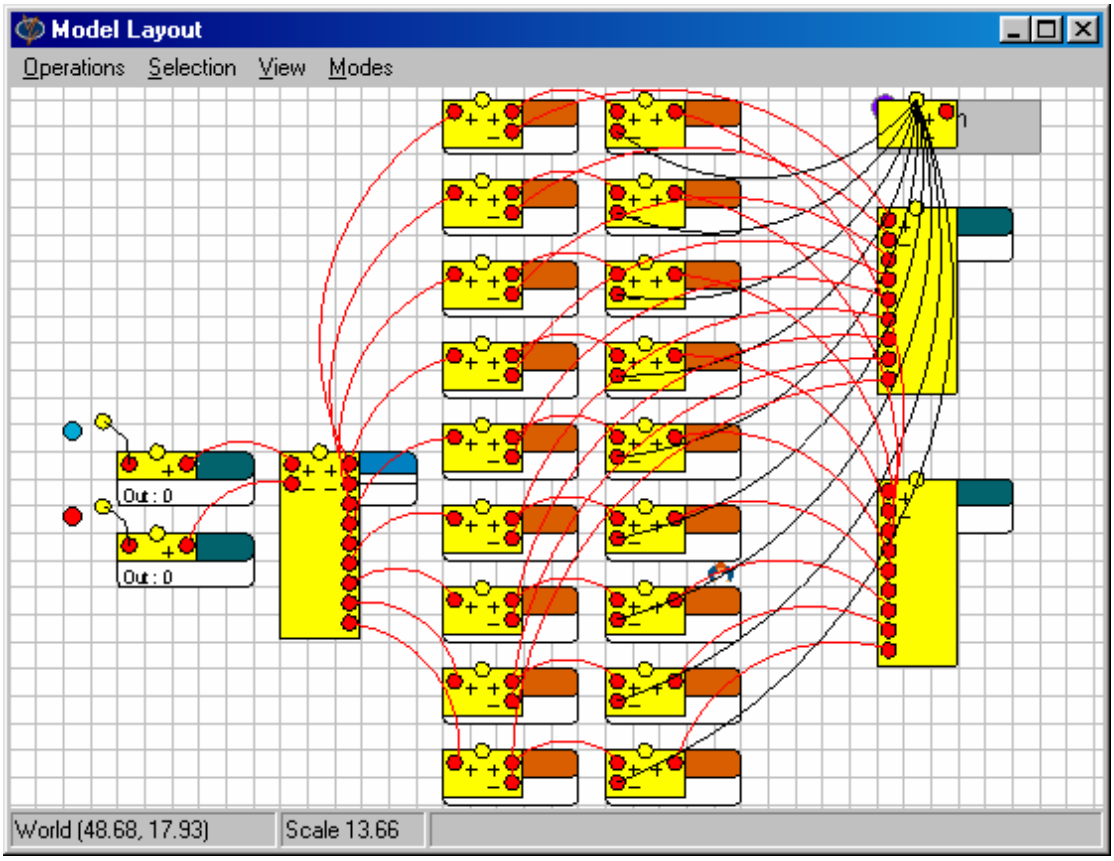
Central Dispatch Location:

Cycletime: `uniform(mins(30),mins(40))`

此处使用多处理器原子来实现。可以同时处理多个进程。为了解决管理人员的需要多少个位置的问题，需要对多处理器的容量进行修改，以找到最合适的容量。

方案 3　Pier3.mod

在本模型中，每个门被拆分成了两部分，第一部分用来进行常规的等待。第二部分用来处理额外发生问题的飞机。显然，这两部分是属于同一个门的，但是建模中，需要使用两个单独的 server 原子。同样还需要 Team 和 operator 原子来代表顾客服务团队。



Team　顾客服务团队。

Operator 服务人员

Gate1_D 到 Gate9_D:

Trigger on entry: `do(closeinput(in(1,c)),calloperators(in(2,c),1))`

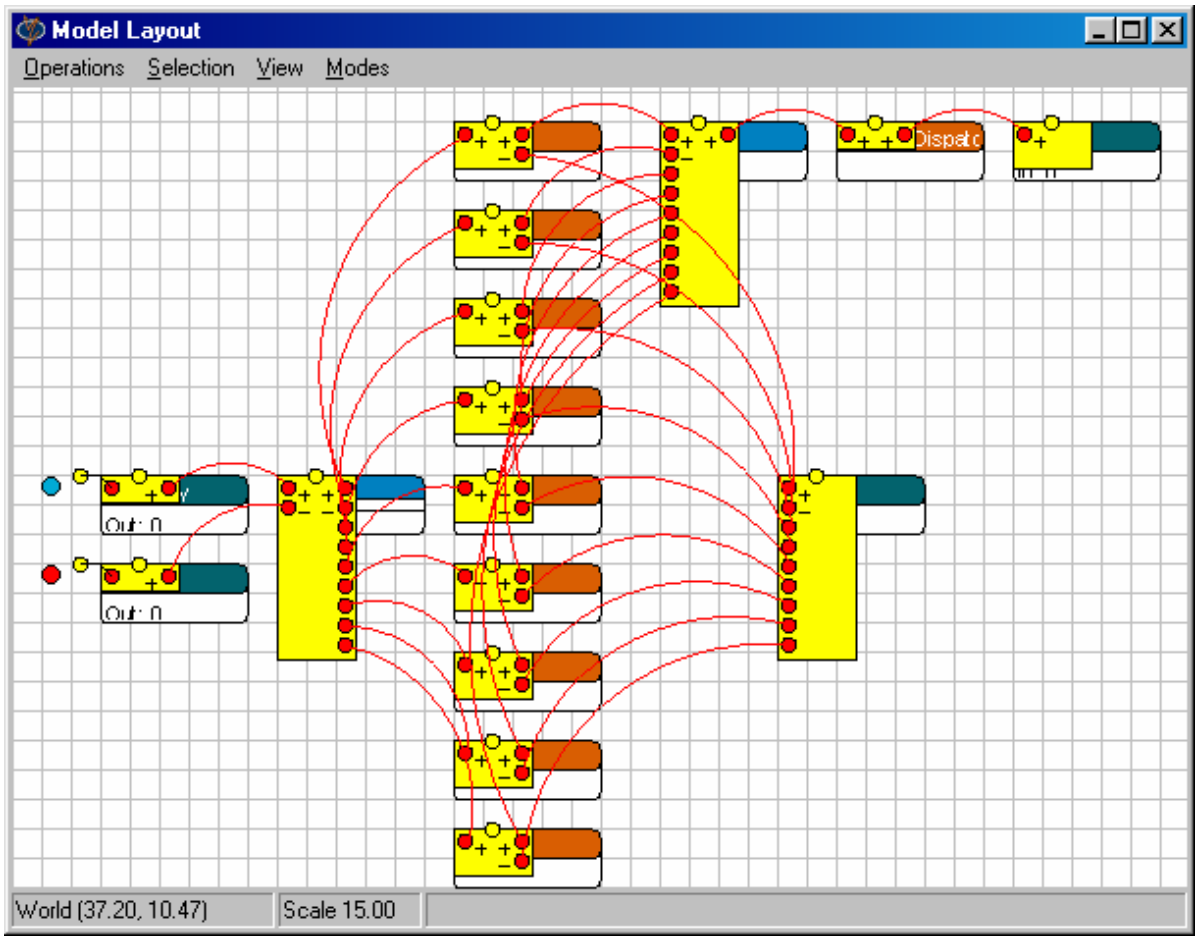
Cycletime: `uniform(mins(10),mins(20))`

Trigger on exit: `do(openinput(in(1,c)),freeoperators(in(2,c),i))`

Experiment Wizard　　实验

PFM3 门的平均使用率。由于门被分为了两部分，所以先分别测量其使用率，再得出门的使用率。

方案 4 Pier4.mod



Central Dispatch Location: Cycletime: `uniform(mins(10),mins(20))`

Queue2：队列代表在中央处理区域中的位置。