[19]中华人民共和国专利局

[11] 公开号 CN 1113582A

[12] 发明专利申请公开说明书

[21]申请号 94117568.5

[51]Int.Cl[6]
G06F 17/00

[43]公开日 1995 年 12 月 20 日

[22]申请日 94.10.20

[30]优先权

[32]93.10.22[33]US[31]141,306

[71]申请人 美国电报电话公司

地址 美国纽约

[72]发明人 马拉·J·贝克

斯蒂芬·G·艾克

[74]专利代理机构 中国国际贸易促进委员会专利商标事务所

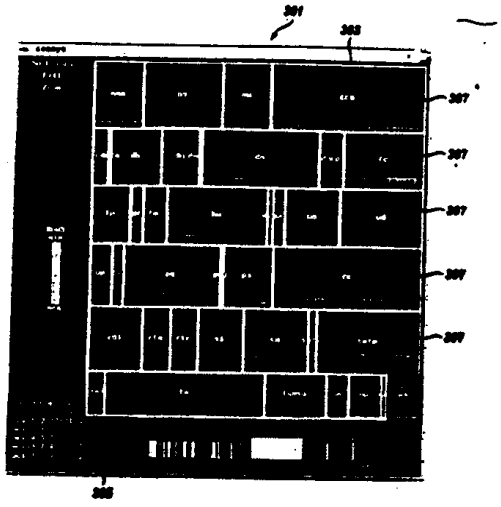
代理人 范本国

说明书页数: 75 附图页数: 11

[54]发明名称 显示大型软件系统分层信息的方法及装置

[57]摘要

对大型软件系统的结构、统计量和特性进行显示的交互的方法和装置。整个软件系统被看成为一个实体。以相似的几何形状给出代码的相似的子划分，该形状具有基本相等的参照框，如基本一致的行高度，所以子划分的相对尺寸能被理解。除尺寸之外，可能使用不同的阴影和着色方式显示改变、软件错误和软件修复。再者，如果软件系统的各种发行版本的数据可用，则可动画显示系统各发行版本的演化，使大型软件系统的历史更易理解。



权 利 要 求 书

1. 在计算机系统的显示中显示与适用统计信息的实体的分级结构有关的至少一个统计量的装置，包括：

显示中的第一区，与分级结构中最高级的一个实体相应；以及

包含在第一区中的多个被包含的区，排列成至少一行，每行高度基本上相同，每个被包含的区与最高级的一个实体之外的实体相应，每个被包含的区的面积代表相应实体的统计量，每个被包含的区被包含在与分级结构中下一最高级中的一个实体相应的区中，且直接包含在包含区中的被包含的区具有形状，以便直接被包含的区的相对尺寸易于从显示中看出。

2. 根据权利要求 1 的装置，其中：

该装置显示适用于实体的另一统计量；以及

与分级结构的给定级中的实体相应的被包含的每个区，根据该区相应的实体的另一统计量的值，被直观地分成子区。

3. 根据权利要求 2 的装置，其中：

所有的区属于同类别的多边形。

4. 根据权利要求 3 的装置，其中：

多边形的类别是矩形。

5. 根据权利要求 4 的装置，其中：

每个被包含的区分别具有一个标识；以及

每个标识至少部分上位于与其实体相应的区内。

6. 根据权利要求 1 的装置，还包括：

显示中的第二区，含有一个交互控制区，用于调节所述第一区被分成的行数。

7. 根据权利要求 1 的装置，还包括：

用于调节每行高度的装置，以便每行高度基本相同，且每行以整数个几何形状充满。

8. 在计算机系统的显示中显示与适用统计信息的实体的分级结构有关的至少一个统计量的方法，包括步骤：

在显示中分配一个第一区，它与分级结构中最高级的一个实体相应；以及

将第一区分成多个含在第一区中的被包含的区，它们排列成至少一行，每行高度基本上相同，每个被包含的区与最高级的一个实体之外的实体相应，每个被包含的区的面积代表相应实体的统计量，每个被包含的区被包含在与分级结构中下一最高级中的一个实体相应的区中，且直接包含在包含区中的被包含的区具有形状，以便直接被包含的区的相对尺寸易于从显示中看出。

9. 根据权利要求 8 的方法，还包括步骤：

分别确定每个被包含的实体的一个标识；以及

显示每个标识，使每个标识的中心分别位于与其被包含的实

体相应的区内。

10. 显示具有多个子系统的软件系统的信息的方法，包括步骤：

确定软件系统的源代码总量；

确定多个子系统中每个子系统各自的源代码量；

将一个显示分成多个矩形区，每个矩形区代表多个子系统中各个子系统；

确定每个矩形区的尺寸，使得代表子系统的每个矩形区具有第一尺寸和一面积，该第一尺寸与代表子系统的所有其他矩形区的尺寸相同，该面积相对于显示的比例与各子系统中源代码量相对于软件系统的源代码总量的比例相同。

11. 根据权利要求 10 的方法，还包括步骤：

分别为每个子系统确定一个名字；以及

在代表每子系统的矩形区内至少部分地显示其名字。

12. 根据权利要求 11 的方法，还包括步骤：

为每个子系统确定一个已修改的源代码量对源代码总量的比率，和一个未修改的源代码量对源代码总量的比率；以及

在每个矩形区中显示第一颜色的区，与其相应区的尺寸的比例与已修改的源代码量对源代码总量的比率相同，显示第二颜色的区，与其相应区的尺寸的比例与未修改的源代码量对源代码总量的比率相同。

13. 根据权利要求 11 的方法，还包括步骤：

为每个子系统确定一个已修改的源代码量对源代码总量的比率，和一个未修改的源代码量对源代码总量的比率；以及

在每个矩形区中显示第一阴影区，与其相应区的尺寸的比例与已修改的源代码量对源代码总量的比率相同，显示第二阴影区，与其相应区的尺寸的比例与未修改的源代码量对源代码总量的比率相同。

14. 显示具有多个发行版本并具有多个子系统的软件系统的信息的方法，包括步骤：

为多个子系统中的每个子系统确定所述多个发行版本中任何版本的各最大源代码量；

通过将所述子系统最大源代码量相加，确定软件系统的最大源代码总量；

将显示分成多个矩形区，每个矩形区代表各子系统最大源代码量；

确定每个矩形区的尺寸，使得代表子系统的每个矩形区具有第一尺寸和一面积，该第一尺寸与代表子系统的所有其他矩形区的尺寸相同，该面积相对于显示的比例与各子系统中最大源代码量相对于软件系统的最大源代码总量的比例相同。

15. 根据权利要求 14 的方法，还包括步骤：

对于各子系统，在每个矩形中为软件系统的每个发行版本显

示一个阴影，它代表所显示版本的源代码量对最大源代码量的比率；以及

为软件系统的每个发行版本，顺序显示所述矩形和所述阴影，所用比率使得阴影的相对尺寸的变化易于从显示中看出。

16. 根据权利要求 14 的方法，还包括步骤：

对于各子系统，在每个矩形中为软件系统的每个发行版本显示一个有颜色的区，它代表所显示版本的源代码量对最大源代码量的比率；以及

为软件系统的每个发行版本，顺序显示所述矩形和所述有颜色的区，所用比率使得阴影的相对尺寸的变化易于从显示中看出。

17. 根据权利要求 16 的方法，还包括步骤：

显示每个有颜色的区的中间尺寸，它从一个发行版本到另一个发行版本改变尺寸，以动画显示有颜色的区的相对尺寸的变化，有助于感受软件系统的演化。

18. 根据权利要求 17 的方法，还包括步骤：

根据用户选择，调节动画比率。

说　　明　　书

显示大型软件系统分层信息的方法及装置

本发明涉及计算机系统，更具体地涉及图形显示大型软件系统的关键特性的方法及装置。

开发一百万行以上源代码的大型软件系统，需要许多训练有素的专业人员付出长时间的艰苦努力。软件系统具有一个总体结构，正如一座大型建筑物具有一个总体结构一样。在大型软件系统的总体结构内，有子系统，目录和文件，它们提供系统的特定功能，正如大型建筑物的层楼、边房和房间提供建筑物的功能一样。如果功能需要演化，则系统的子系统、目录和文件也可另外演化，正如当建筑物需求改变时，可以增加、去除或修复一个建筑物的边房和房间。大型建筑物具有其原始结构，显示于一系列称为篮图的易于形象化的技术说明中。为助于以后对建筑物的建造和修复，对篮图进行修改，使得原始结构和构造以及以后对原始结构和构造所做的任何改变都以视觉上可辨认并易于理解的方式显示出来。不幸的是，对于含有子系统、目录和文件的大型软件系统的原始结构和构造，以及以后的改变，以视觉可识别并易于理解的方式进行显示的方法，当前并不存在。

所知的小型软件系统的显示和显象技术不能成攻地扩大到用于大型软件系统。大型软件系统的源代码列表具有太多的细节，而难于理解。流程图和结构框图不仅变得具有过多细节，而且后来对软件系统的改变也可能放弃了与系统的当前版本毫不相关的原始流程、结构和抽象用语。另外，目前出现的大型系统的显示和显象技术，如91年《显象学报》(*Proceedings Visualization*)中，*Johnson* 和 *Shneiderman* 在文章"三图：显象分层信息结构的一种空间填充方法"中所提出的三图(*three—map*)技术，产生难于理解的显示。该三图技术用矩形填充显示区域，矩形的受控量是其面积。仅控制矩形面积造成某些矩形长且窄，某些短且宽，还有接近正方形的，但具有相同的面积。这使得结构细节难于显象，因为观者必须比较许多具有不同长度和宽度的矩形的面积，以做相应的判断。这样，在技术中，有必要以可理解的方式，对与大型软件系统的子系统、目录和文件有关的信息和统计量进行图形显示，为以后的软件维护和开发人员所用。

本发明的一个目的，是提供一种显示大型软件系统的结构和构造的装置，使得技术人员能够理解其子系统、目录和文件之间的关系。

本发明的另一个目的，是提供一种显示与大型软件系统的代码变化有关的信息和统计量的装置，使得技术人员能够理解该系统、其子系统、目录和文件的功能演化。

本发明的再一个目的，是提供一种显示大型软件系统的结构和构造的方法，使得技术人员能够理解其子系统、目录和文件之间的关系。

本发明还有一个目的，是提供一种显示与大型软件系统的代码变化有关的信息和统计量的方法，以帮助技术人员看到并理解该系统、其子系统、目录和文件的功能的演化。

根据本发明一方面的简述，前述目的通过提供一种对实体分级结构的至少一种统计量进行显示的装置实现，在计算机系统的显示中统计信息适用于该实体。该装置包括显示中的第一区，它与分级结构的最高级的一个实体相应。第一区中所含的是多个被包含的区，排列成至少一行，各行高度基本相同。每个被包含的区与最高级的一个实体之外的实体相应。每个被包含的区的面积代表相应实体的统计量。一个区中所含的每个被包含的区与分级结构的下一最高级中的一个实体相应。直接含在包含区中的被包含的区具有形状，使得直接被包含的区的相对尺寸易于从显示中看出。

根据本发明另一方面，发明目的通过提供一种对实体分级结构的至少一种统计量进行显示的方法实现，在计算机系统的显示中统计信息适用于该实体。该方法在显示中分配第一区，与分级结构中最高级一个实体相应；并且将第一区分为多个含在第一区中的被包含的区，排列为至少一行，每行高度基本相同，每个被包含的区与除最高级的一个实体之外的实体相应，每个被包含的区的面积代表相

应实体的统计量，一个区中所含的每个被包含的区与分级结构的下一最高级中的一个实体相应，并且直接含在包含区中的被包含的区具有形状，使得直接被包含的区的相对尺寸易于从显示中看出。

尽管说明书得出所附的对本发明主题进行了具体明确说明的权利要求书，但通过以下描述并结合附图，本发明将更好理解。

图 1 是显示分层软件系统的计算机系统例的框图。

图 2 是显示装置和方法的读关键特性部分的流程图。

图 3 是以多行显示子系统相对尺寸和新开发代码比例的显示画面。

图 3*A* 是单行显示与图 3 所示相同信息的显示画面。

图 4 是最佳实施方式中显示每个子系统的相对尺寸、新开发代码比例以及目录细节的显示画面。

图 5 是图 4 中一个子系统的经放大至充满显示屏的画面。

图 6 是最佳实施方式中显示每个子系统中与新的、非注释源代码行和软件错误修复有关的子系统尺寸的显示画面。

图 7 是显示与用于软件错误修复的新的、非注释源代码行以用为修复先前软件错误中的错误而需的软件错误修复数目有关的子系统尺寸的显示画面。

图 8、9 和 10 是最佳实施方式中显示子系统尺寸关于时间(即发行版本)而变化的显示屏幕的画面。

参照图 1，计算机系统 101 示于框图中。计算机系统 101 含有

终端103，它向操作计算机系统101的技术人员提供输出并从之接收输入，处理机113，它执行实际操作，存储器115，它含有处理机113所执行的程序117，以及数据区119，它含有与分层软件系统有关的数据，也可含有分层软件系统本身。计算机系统101还具有一个海量存储部件120，用于存储大型程序和大量数据。

更具体些，终端103含有一个显示屏105，处理机113在显示屏上向操作员显示信息。显示屏105还含有一个指示箭头107，它确定显示屏105上的一个位置，并可在键盘109或鼠标器111的控制下移动。操作员通过从键盘109和/或鼠标器111输入对计算机系统101的运行进行控制。处理机113可以是任何种类的处理机，从个人计算机到巨型计算机。存储器115可以含有程序和数据存储用的随机存取存储器和只读存储器。海量存储部件120可以含有一个磁盘、一个光盘，或者甚至一个远程数据库。

当使用计算机系统101处理并显示大型分层软件系统的信息，即统计量时，操作员利用键盘109或鼠标器111输入一条或多条命令；处理机113如所需执行程序117，以执行命令，并在显示屏105上显示结果。然后操作员能够利用键盘109和/或鼠标器111发布一条或多条命令，以便更详细地检查或处理该结果。

现在参照图1和2，一个软件系统的名称已经在键盘109上键入或通过鼠标器111选择，以便开始对所选的软件系统的特性和结构进行显示。与关键特性有关的统计量关联于所选的软件系统，并

且将这些统计量读入存储器115，若不在其中，则加以存储以备本发明的显示程序122使用。然后，处理机113处理软件系统显示程序122，以提供显示，有助于操作员更容易地理解软件系统的特性和结构。附录*A*中给出*C*语言编程的显示程序122的列表。

处理机113读入软件系统的一种发行版本(如最新版本)的非注释源代码行(*NCS*行)的数目，以及一个或多个先前版本的*NCS*行数目。*NCS*行数作为许多显示的基线，如下所述。技术人员将认识到能够使用其他基线单位，如源代码量，而不是*NCS*行数量。这样小的修改是在本发明的范围之内的。

参照图2，显示出由处理机113读入的子系统特性和统计量。每个子系统的名字与其发行标志和*NCS*数一起读入。另外，对于一个发行标志的每个子系统，读入新*NCS*行数，各子系统内的每个目录的名字，各目录内的每个文件的名字，每个目录内*NCS*行数，以及每个目录文件的新*NCS*行数。再者，也读入每个目录的软件错误修正的数目和对先前软件修正的软件错误修正数目。将统计量读入存储器115，可以对这些统计量进行快速访问，所以操作员能够使用本发明的交互和/或动画特性，而没有因经常从诸如磁盘驱动器那样的慢速设备访问数据而造成延迟。

现在参照图3，将解释对大型分层软件系统的统计量和特性的显示。首先，在显示屏301上定义一个系统显示空间303，在其中显示子系统统计量和特性。显示屏301也具有一个控制和指示空间

305，位于显示空间 303 的左下方。显示空间 303 是一个交互窗口，在指示箭头和鼠标器控制下，可以在 *x* 和 *y* 方向变化尺寸。显示空间 303 是矩形的，这是最佳的，因为矩形表示是易于直观地理解的形式，尽管可以使用其他几何形状，如椭圆或圆，它们在本发明的范围内。

显示空间 303 被分为数个几何形状 307，每个几何形状的尺寸，即面积代表各子系统的 *NCS* 行数。几何形状 307 是矩形的，但是可以使用其他几何形状，如三角形、梯形等。尤其如果使用圆形显示空间，则使用同心相临环，而不是这些环的弧和段作为几何形状代表每个子系统。

将显示空间 303 分为一行或多行几何形状 307。每行在 *y* 方向上的延伸(即高度)基本上相同。如只有一行，如图 3*A* 所示，则所有形状在 *y* 方向上具有相同的延伸。这造成很高很窄的几何形状，难于直观地解释各空间之间的尺寸关系，仅具有统一的高度。因为很高很窄的几何形状难于直观地解释，则所以设置一个交互式行控制滑动区 309。利用指示箭头 107 和鼠标器 111 将指示条从 *MIN* 拖到 *MAX*，使显示区 303 中的行数从 1 增加到软件系统中子系统的数目作为最大数目。操作员可以选择其易于解释的行数。图 3 所示为 6 行。

子系统在每行中从左到右以降字母序排列。由于次序是由名字而不是尺寸确定的，所以显示每个几何形状 307 的方法也含有对不

便的子系统尺寸的解决方案，如可能产生的在行尾出现非常大的文件。该方法取子系统数 *Subs*，并除以行数 *Rows*，取子系统的 *NCS* 数 *NCSL*，并除以行数。显示程序 122 在第一行中以字母序分配待显示的子系统。显示程序 122 试探性地将(*Subs*/*Rows*)的商作为第一行中的子系统数。然后，显示程序 122 将第一行中所分配的子系统的 *NCS* 行数相加，并与(*NCSL*)/(*Rows*)的商相比。如果试探性分配的子系统的 *NCS* 行数小，则显示系统可以在该行中分配一个附加的子系统。一旦第一行分配完毕，则调节该行的高度，使得其面积与显示空间 303 的总面积的比值，与第一行中的子系统的 *NCS* 行数与所有子系统的 *NCS* 行总数的比值相同。该方法以同样方式处理后续行。因为每行的面积被调节为显示空间 303 的相应部分，则始终有用于剩余子系统的空间。这样，本发明的方案在于对高度进行稍许改变，或者在一行为所有子系统提供较多的面积，以便得到该行中的最后一个文件，或者提供较少的面积，并将行尾的最后一个文件强制到下一行首。这种行高自调节使得可以不必使用不规则的行长，或者不必在几何形状中间断开，以使填充显示空间 303 中不需要的行。该方案提供了显示空间 303 的百分之百的利用。

如上所述，每个几何形状 307 的面积对整个显示空间 303 的比例，与其相应子系统的 *NCS* 行数对整个软件系统的 *NCS* 行总数的比例相同。某些几何形状 307 被轻微地遮盖了。图 3 中，阴影的这种对比代表子系统中新编写的 *NCS* 行的比例。这样，图 3 显示了两

种统计量，每个子系统的相对尺寸和相对于每个子系统尺寸的新代码量。如图 3*A* 所示，单行显示 303*A*，对于显示每个子系统中新代码量是有用的，尽管难于对各子系统的相对尺寸进行解释。子系统中缺少新代码可能指示着增加新功能的地方。

图 3*A* 中，几何形状 307*A* 具有代表非新 *NCS* 行量的具有灰度的阴影。将该阴影改变到不同的灰度，指示各子系统的尺寸。在具有 256 或更多灰度级的灰度级计算机显示器上，这是容易提供的。可选择地，对于具有彩色显示的系统，可以使用色彩。对比色可以有助于操作员看到正被分析的系统。例如，对于彩色显示，形状 307*A* 中的非新 *NCS* 行可以用一个色彩表示，如深篮色，而形状 307*A* 中的新 *NCS* 行可以用灰色、较浅的篮色或其他对比色表示。因为色彩区分对于眼睛易于鉴别，本发明的最佳实施方式采用彩色显示，对于具有彩色显示的系统，附录 *A* 是一致的。

现在参照图 4，可以显示软件系统的进一步的细节。显示空间 403 基本上与显示空间 303 相同，除代表一个子系统的每个几何形状 407 还具有显示于其内的相应子系统的目录。每个目录用一个几何形状表示，其尺寸，即面积，对其子系统几何形状 407 的面积的比率，与目录的 *NCS* 行数对其相应子系统的 *NCS* 行数的比率相同。以字母序在各子系统中排列目录。每个目录的统计量和特性也得以显示。通常统计量和特性以及目录名字在显示 403 中的不可读的。鉴于此，控制空间 405 具有交互式信息区 410，逻辑上链接于指示

箭头107的位置。信息区410给出指示箭头107所在处的子系统的*NCS*行数。该信息由滑块上的指示器的位置以图形方式给出，并由信息区410中的数字表示给出。同样，信息区410在一字母数字表示中指明指示箭头107所在处的子系统的名字。控制空间405具有变焦交互区412，并且如果指示箭头107点亮一个如子系统*w*1的子系统时，指示箭头位于变焦区412上，且由鼠标器111选择变焦特性，显示程序在前景中从显示空间403打开一个窗口。控制空间405的最佳实施方式也是具有多种可用色彩的显示空间。

现在参照图5，显示了一个所选子系统*w*1的窗口501。该实施方式中，代表子系统*w*1的几何形状507是矩形的，几何形状507被分成矩形509，所有矩形509在*y*方向上具有相同的延伸。所以，每个目录矩形509的相对尺寸，取决于其在*x*方向上的延伸。每个目录的名字大致位于各矩形的*x*方向的中心，但如果在相邻目录矩形中有可用空间，则可以伸到边界外。为避免冲突，目录的对中的名字在正*x*和*y*方向以斜线序排列。

窗口501中显示了三种阴影。浅灰520和黑522提供一个背景，它代表几何形状507所代表的整个子系统*w*1的代码的新*NCS*行的百分比(浅灰)和代码的先有*NCS*行的百分比(深灰或黑)。在窗口501的前景中的中灰524，代表每个目录509中新*NCS*行的百分比。这样，在窗口501中，所选子系统中的新*NCS*行的百分比，组成子系统的目录的相对尺寸，以及子系统中的每个目录的修改的

NCS 行量，所有这些都易于视觉比较。

在显示 301 的前景中可变焦打开多于一个的窗口。利用多窗口帮助操作员，每个窗口由指示箭头 107 的位置逻辑上键接到信息区 410，以使数字和字母区显示指示箭头 107 所在处的子系统，甚至在子系统的变焦窗口中。当操作员完成一个窗口时，如窗口 501，可以在键盘 109 上键入适当命令，或用鼠标器 111 激活相同命令，关闭窗口。

现在参照图 6，描述另一重要统计量的显示。显示空间 603 与图 4 中的显示 403 相同，都具有代表当前发行版本的子系统及其当前目目录的相对尺寸的几何形状。然而，在显示空间 603 中，每个几何形状 607 的尺寸代表子系统中新 *NCS* 行数对软件系统中新 *NCS* 行数的比率，每个目录的尺寸代表其相应子系统的新 *NCS* 行数。再者，每个几何形状 607 中的对比阴影，用一个灰色阴影代表为修正软件错误而在每个子系统中包括的新 *NCS* 行的百分比，用不同的灰色阴影代表为修正一个软件错误或多个软件错误而在每个目录中所包括的新 *NCS* 行的百分比。这样，显示空间 603 指明了子系统和目录中在所显示的发行版本之前为补救已知的软件错误而进行的开发所在。也将显示为增加功能而非修正错误而进行新开发处的面积，因为那些几何形状 607 尺寸太大而阴影少。如图 4 和 5 所示，如果采用彩色显示，则这些细节最好以对比色显示，而不用图 6 所用的各种灰色阴影。

图 7 更具体地显示了为修正软件错误而改变 *NCS* 行。图 7 中，每个几何形状 707 的尺寸代表在最新软件系统版本的相应子系统中发现并修正的的软件错误数量。根据由最新发行版本发现和修正的软件错误的数量，确定目录的尺寸。图 7 的阴影代表所做的软件错误修正与先前软件错误修正的百分比。许多修正又修正的几何形状 707，例如，子系统 730，指明有故障的目录或子系统，可能最好进行重写指令并重写。

现在参照图 8、9 和 10，解释对两个更多发行版本的软件系统的演化直观化的方法。处理机 113 读取所有版本的所有子系统名字和统计量，作为显示的部分。在读过程中，处理机 113 在存储器系统 115 中，建立所考虑的软件系统版本的任何一个中的存在的每个子系统的列表，以及每个子系统的最大尺寸。一旦积累起该数据，则显示空间 803、903 和 1003(这是在显示屏 301 上的时间顺序)显示矩形 807、907 和 1007 所填充的行，矩形代表软件系统中所存在的每个子系统，它们的尺寸根据每个子系统中存在的最大 *NCS* 行数而定。因为这些统计量实际上是不变的，所以显示空间 803、903 和 1003 具有相同的子系统和相同的子系统矩形尺寸。

软件系统的演化由代表子系统的最大尺寸的矩形 807、907 和 1007 内的阴影尺寸变化显示。对每个子系统，阴影代表软件发行版本之一中的子系统的相对于子系统最大尺寸的尺寸。

图 8 的显示空间 803 显示了软件系统的最早版本的矩形 807

的阴影。图 10 的显示空间 1003 显示了较后时间的软件系统较后版本的矩形 1007 的阴影。交互式速度滑动器 832 允许操作员在鼠标器控制下选择显示空间 803 中的显示与显示空间 1003 中的显示之间的时间量。可选择地，帧滑动器 834 允许操作员通过指示箭头 107 和鼠标器 111 的控制选择图 8 所示第一和图 10 所示第一百零五之间的任何显示。

图 9 显示了图 8 所表示的最早版本与图 10 所表示的当前版本之间的过渡。图 9 可以表示最早和当前发行版本之间的一种实际软件版本，或者它可以表示两种发行版本之间的代码的增加。这种增加的目的在于在那些例子中增加向操作员提供的动画的平滑性，实际发行数据不足以提供软件子系统演化的平滑动画的显示。

bottom.C

```
1   #include <assert.h>
2   #include <iostream.h>
3   #include <memory.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <string.h>
7
8   #include <X11/Intrinsic.h>
9   #include <X11/StringDefs.h>
10  #include <Xm/MainW.h>
11  #include <Xm/RowColumn.h>
12  #include <Xm/PushB.h>
13  #include <Xm/CascadeB.h>
14  #include <Xm/Frame.h>
15  #include <Xm/Scale.h>
16  #include <Xm/Label.h>
17  #include <Xm/Form.h>
18  #include <X11/Xirisw/GlxMDraw.h>
19
20  #include <gl/gl.h>
21
22  #include "bottom.h"
23  #include "seesys.h"
24
25  #define char_h          .02
26
27  #ifndef ABS
28  #       define ABS(x)   ((x)>0 ? (x) : -(x))
29  #endif
30
31
32  void
33  Bottom :: drawslider(int partitions)
34
35          double percentfill;
36          double x1, x2, y1, y2;
37          x1 = .2; x2 = .8; y1 = .3; y2 = .7;
38
39          // colored partitions
40          double width = (x2 - x1) / partitions;
41          for (int i=0; i < partitions; i++) {
42                  color(cols[i]);
43                  FillRect(x1 + i*width, y1, width, y2-y1);
44          }
45
46          // bounding rectangle
47          color(MY_WHITE);
48          FrameRect(x1, y1, x2-x1, y2-y1);
49
50          // draw subsystem bars in slider
51          double offset;
52          if (!do_modules) {
53                  linewidth(3);
54                  color(MY_WHITE);
55                  for (i=0; i < nss; i++) {
56                          if (nframes > 1)
```

```
                        offset = x1 + ssvar[i]/maxssframevar * (x2-x1);
                    else
                        offset = x1 + ssvar[i]/maxssvar * (x2-x1);
                    DrawLine(offset,y1,offset,y1+.75*(y2-y1));
                    barx[i]=offset;


                if (active_ss >=0 ){
                        color(DK_RED);
                        double x = barx[active_ss],
                        y = y1+1.25*(y2-y1);
                        if (ssvar[active_ss] == 0)
                                percentfill = 0;
                        else
                                percentfill = ssfill[active_ss] /
                                        ssvar[active_ss] * 100;
                        DrawLine(x,y1,x,y);
                        char buf[MAX_STRING];
                        if (do_fill[active_ssvar] == 1)
                          sprintf(buf, " %s: %.0f (%.0f%%)", ssnames[active_ss],
                                ssvar[active_ss], percentfill);
                        else
                          sprintf(buf, " %s: %.0f", ssnames[active_ss],
                                ssvar[active_ss]);
                        DrawString(x,y,buf);
                }
                linewidth(1);
        }
        else {  // for modules
                linewidth(3);
                for (i=0; i < nmod; i++) {
                        if (nframes > 1)
                          offset = x1 + modvar[i]/maxmodframevar * (x2-x1);
                        else
                          offset = x1 + modvar[i]/maxmodvar * (x2-x1);
                        barx[i]=offset;
                }
                if (active_module >= 0) {
                        color(DK_RED);
                        double x = barx[active_module],
                        y = y1+1.25*(y2-y1);
                        if (modvar[active_module] == 0)
                                percentfill = 0;
                        else
                                percentfill = modfill[active_module] /
                                        modvar[active_module] * 100;
                        DrawLine(x,y1,x,y);
                        char buf[MAX_STRING];
                        if (domod_fill[active_modvar] == 1)
                          sprintf(buf, " %s: %.0f (%.0f%%)",
                                modnames[active_module], modvar[active_module],
                                percentfill);
                        else
                          sprintf(buf, " %s: %.0f", modnames[active_module],
                                modvar[active_module]);
                        DrawString(x,y,buf);
```

```
113                     }
114             }
115
116             // write slider partition labels
117             char buf[MAX_STRING];
118             color(MY_WHITE);
119             for (i=0; i<=PARTITIONS; i++) {
120                     if (do_modules)
121                             sprintf(buf, " %.f", maxmodvar/PARTITIONS * i);
122                     else if (nframes > 1)
123                             sprintf(buf, " %.f", maxssframevar/PARTITIONS * i);
124                     else
125                             sprintf(buf, " %.f", maxssvar/PARTITIONS *i);
126                     DrawString(x1 + i*((x2-x1)/PARTITIONS), y1-7*char_h, buf,
127                             VxDrawCenter);
128             }
129     }
130
131     // this is a kluge
132     static GLXconfig glxConfig [] = {
133         { GLX_NORMAL, GLX_DOUBLE,   TRUE },
134         { GLX_NORMAL, GLX_COLORMAP, TRUE },
135         { GLX_NORMAL, GLX_ZSIZE,    GLX_NOCONFIG },
136         { 0, 0, 0 }
137     };
138
139
140     Bottom :: Bottom(char const *name, Widget parent ) :
141
142             // default constructors
143             VxGLDrawingArea(name, parent, glxConfig),
144             VxMouseable( VxDown | VxDrag | VxMotion ),
145             VxFunctionality(this)
146     {
147             pub.Subscribe(this, ACTIVE_SS);
148             pub.Subscribe(this, ACTIVE_MODVAR);
149             pub.Subscribe(this, ACTIVE_MODULE);
150             pub.Subscribe(this, ACTIVE_SSVAR);
151             pub.Subscribe(this, FRAME_CHANGE);
152     }
153
154     void
155     Bottom :: DoExpose(int, int, int, int)
156     {
157       GLXwinset( MyDisplay(), MyWindow());
158       ortho2(0, 1, 0, 1);
159
160       color(MY_BLACK);
161       clear();
162
163       // draw slider
164       drawslider(PARTITIONS);
165
166       swapbuffers();
167     }
168
```

```
169
170   void
171   Bottom :: DoMouse(VxMouseActions what, int /*button*/,
172                     XEvent const * /*xevent*/)
173   {
174           double wmx, dist;
175           int closest, ss;
176
177           switch(what) {
178           case VxDown:
179           case VxMotion:
180           case VxDrag: {
181                   wmx = XStoU( MouseX() );
182                   dist = ABS(barx[0] - wmx);
183                   closest = 0;
184                   if (do_modules) {
185                           for (int i=0; i<nmod; i++)
186                                   if (ABS(barx[i] - wmx) < dist) {
187                                           dist = ABS(barx[i] - wmx);
.88                                         closest = i;
189                                   }
190                           if (dist < .005) {
191                                   ::active_module = closest;
192                                   int cummods = 0;
193                                   for (i=0; i<nss; i++) {
194                                     if (::active_module >= cummods)
195                                       ss = i;
196                                     cummods += modulesPerSS[i];
197                                   }
198                                   ::active_ss = ss;
199                                   pub.Changed( ACTIVE_MODULE, ::active_module );
200                                   pub.Changed( ACTIVE_SS, ::active_ss );
201                           }
202                   }
203                   else {
204                           for (int i=0; i<nss; i++)
205                                   if (ABS(barx[i] - wmx) < dist) {
206                                           dist = ABS(barx[i] - wmx);
207                                           closest = i;
208                                   }
209                           if (dist < .005) {
210                                   ::active_ss = closest;
211                                   pub.Changed( ACTIVE_SS, ::active_ss );
212                           }
213                   }
214                   break;
?15           }
 16
217           default:
218                   cout << __FILE__ << __LINE__ << "not reached" << endl;
219           }
220   }
221
222   void
223   Bottom :: Update(int article,  int /* part */,  void *)
?24   {
```

```
225         switch (article)
226         case ACTIVE_SS:
227         case ACTIVE_SSVAR:
228         case ACTIVE_MODVAR:
229         case FRAME_CHANGE:
230         case ACTIVE_MODULE:
231                 ExposeMe();
232                 break;
233         default:
234                 cout << __FILE__ << " " << __LINE__ << "ERROR - not reached"
235                         << endl;
236
237
238
```

left.c

```
1   #include <assert.h>
2   #include <iostream.h>
3   #include <memory.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <string.h>
7   #include <math.h>
8
9   #include <X11/Intrinsic.h>
10  #include <X11/StringDefs.h>
11  #include <Xm/MainW.h>
12  #include <Xm/RowColumn.h>
13  #include <Xm/PushB.h>
14  #include <Xm/CascadeB.h>
15  #include <Xm/Frame.h>
16  #include <Xm/Scale.h>
17  #include <Xm/Label.h>
18  #include <Xm/Form.h>
19  #include <X11/Xirisw/GlxMDraw.h>
20
21  #include <gl/gl.h>
22
23  #include "left.h"
24  #include "seesys.h"
25
26  // *********** stats ***************
27  #define VARWIDTH        .95
28  #define VARX            .05
29  #define char_h          .02
30
31  // stats
32  #define STATY            .15
33  #define STATHEIGHT       (nssvars * char_h)
34  #define MODSTATY         .30
35  #define MODSTATHEIGHT    (nmodvars * char_h)
36
37
38  // what has been selected in identify
39  enum {
40          TOGGLE = 1,
41          SLIDER = 2
42  };
43
44
45  Toggle :: Toggle(double x_a, double y_a, int *flag_a,
46                   char *offlabel, char *onlabel, int mypub_a)
47  {
48          x = x_a;
49          y = y_a;
50          flag = flag_a;
51          labels[0] = offlabel;
52          labels[1] = onlabel;
53          mypub = mypub_a;
54
55
56
```

```
57  Slider :: Slider(double x_a, double y_a, double h_a, double w_a,
58            int min_a, int max_a, int *bar_a, int mypub_a,
59            char *top_label_a,
60            char *left_toplabel_a, char *left_bottomlabel_a,
61            char **barlabels_a)
62  {
63    x = x_a;
64    y = y_a;
65    h = h_a;
66    w = w_a;
67
68    min = min_a;
69    max = max_a;
70    bar = bar_a;
71
72    mypub = mypub_a;
73
74    top_label = top_label_a;
75    left_toplabel = left_toplabel_a;
76    left_bottomlabel = left_bottomlabel_a;
77
78    barlabels = barlabels_a;
79  }
80
81
82
83  void
84  Left :: drawslider(Slider *s)
85  {
86    char buf[MAX_STRING];
87    double extend = .15 * s->w;
88
89    color(MY_CYAN);
90    FillRect(s->x, s->y, s->w, s->h);
91    LineWidth(3);
92
93    color(DK_YELLOW);
94    double bary = s->y + s->h * *s->bar/(s->max - s->min);
95
96    // slider bar
97    DrawLine(s->x - extend, bary, s->x + s->w+extend, bary );
98    if( s->barlabels )
99      strcpy( buf, s->barlabels[ *s->bar ] );
100   else
101     sprintf( buf, "%d", *s->bar );
102   DrawString(s->x+s->w + 2*extend, bary + .25*char_h,
103                buf, VxDrawLeft);
 04
105   color(MY_WHITE);
106   DrawString(s->x + .5*s->w, s->y + s->h+.45*char_h,
107              s->top_label, VxDrawCenter);
108   LineWidth(1);
109 }
110
111 int
112 Left :: identifyslider(double wmx, double wmy, Slider *s)
```

```
113   {
114     if( s->x<=wmx && wmx<=s->x+s->w
115        && s->y<=wmy && wmy<=s->y+s->h )
116       {
117         *s->bar = (int)( (wmy-s->y)/s->h  *
118                         (s->max-s->min) + s->min + .5);
119         return SLIDER;
120       }
121     return 0;
122   }
123
124   // this stuff should go!
125   // ********** frame vars **********
126   #define framex .55
127   #define framew .65
128   #define framey .65
129   #define frameh .8
130   #define extend (.15 * (framew-framex))
131   #define rowy .4
.32   #define rowh .55
133
134   /* right justified label */
135   double
136   Left :: windsor(double low, double x, double high)
137   {
138     if(x<low) return low;
139     if(x>high) return high;
140     return x;
141   }
142
143
144
145   // drawssstats - shows subsystem statistics
.46   void
147   Left :: drawssstats()
148   {
149     int i;
150     Colorindex c;
151
152     /* draw background rectangle */
153     color(MY_GRAY);
154     rectf( VARX,            STATY,
155            VARX+VARWIDTH,   STATY-STATHEIGHT);
156
157     color(MY_WHITE);
158     DrawString(VARX+.5*VARWIDTH, STATY+.25*char_h,"SUBSYSTEM STATS",VxDrawCenter);
159
 60
161     /* print the statistic names */
162     for(i=0; i<nssvars; i++)
163       {
164         if( i==active_ssvar )
165           color(DK_BLUE);
166         else
167           color(MY_GRAY);
168
```

```
169         rectf( VARX,            STATY - i*char_h,
170                VARX+VARWIDTH,   STATY - (i+1)*char_h);
171
172         /* black line between variables */
173         color(MY_BLACK);
174         move2( VARX,           STATY - i*char_h );
175         draw2( VARX+VARWIDTH, STATY - i*char_h );
176
177         c = i==active_ssvar ? DK_YELLOW : MY_WHITE;
178         color(c);
179         DrawString(VARX+.5*VARWIDTH, STATY - (i+.75)*char_h, ssvarnames[i],VxDrawRigt
180         if (active_ss >= 0) {
181           char buf[MAX_STRING];
182           sprintf(buf, " %.f", get_ssvar(i,active_ss));
183           DrawString(VARX+.5*VARWIDTH,STATY - (i+.75)*char_h, buf, VxDrawLeft);
184
185
186
187
188
189   // drawmodstats - shows module statistics
190   void
191   Left :: drawmodstats()
192   {
193     int i;
194     Colorindex c;
195
196     /* draw background rectangle */
197     color(MY_GRAY);
198     rectf( VARX,            MODSTATY,
199            VARX+VARWIDTH,   MODSTATY-MODSTATHEIGHT);
200
201     color(MY_WHITE);
202     DrawString(VARX+.5*VARWIDTH, MODSTATY+.25*char_h,"MODULE STATS",VxDrawCenter);
203
204     /* print the statistic names */
205     for(i=0; i<nmodvars; i++)
206       {
207         if( i==active_modvar )
208           color(DK_BLUE);
209         else
210           color(MY_GRAY);
211
212         rectf( VARX,            MODSTATY - i*char_h,
213                VARX+VARWIDTH,   MODSTATY - (i+1)*char_h);
214
215         /* black line between variables */
216         color(MY_BLACK);
217         move2( VARX,           MODSTATY - i*char_h );
218         draw2( VARX+VARWIDTH, MODSTATY - i*char_h );
219
220         c = i==active_modvar ? DK_YELLOW : MY_WHITE;
221         color(c);
222         DrawString(VARX+.5*VARWIDTH, MODSTATY - (i+.75)*char_h, modvarnames[i],
223           VxDrawRight);
224         if (active_module >= 0) {
```

```
225           char buf[MAX_STRING];
226           sprintf(buf, " %.f", get_modvar(i,active_module));
227           DrawString(VARX+.5*VARWIDTH,MODSTATY - (i+.75)*char_h, buf, VxDrawLeft);
228
229
230
231
232   // identifymodstat - see if a different module statistic was chosen
233   void
234   Left :: identifymodstat(double wmx, double wmy, int *statbutton)
235   {
236     *statbutton = -1;
237
238     // identify button
239     for(int i=0; i<nmodvars; i++)
240       if(( VARX <= wmx && wmx <= VARX+VARWIDTH )
241           &&
242          (MODSTATY-(i+1)*char_h <= wmy && wmy <= MODSTATY-i*char_h ))
243         {
244           *statbutton = i;
245           break;
246         }
247       else
248           *statbutton = active_modvar;
249   }
250
251
252   // set_ssstats - calculates subsystem stats from module stats
253   void
254   Left :: set_ssstats()
255   {
256           double sstot;
257           int i, j, k, offset=0;
258
259           nssvars = nmodvars;
260           for (i=0; i<nmodvars; i++) {
261                   ssvarnames[i] = modvarnames[i];
262                   offset = i * nframes * nmod;
263                   for (j=0; j<nss; j++) {
264                           sstot = 0;
265                           for (k=0; k<modulesPerSS[j]; k++, offset++)
266                                   sstot += modvar_all[offset];
267                           ssvar_all[i*nss+j] = sstot;
268                   }
269           }
270   }
271
272
273   // identifyssstat - see if a different subsystem statistic was chosen
274   void
275   Left :: identifyssstat(double wmx, double wmy, int *statbutton)
276   {
277     // identify button
278     for(int i=0; i<nssvars; i++)
279       if( VARX <= wmx && wmx <= VARX+VARWIDTH )
280           &&
```

```
281          (STATY-(i+1)*char_h <= wmy && wmy <= STATY-i*char_h ))
282        {
283          *statbutton = i;
284          return;
285        }
286
287    // did not find
288    *statbutton = active_ssvar;
289  }
290
291
292  // identifyframe - see if a different frame was chosen
293  void
294  Left :: identifyframe(double wmx, double wmy, int *framepos)
295  {
296    // identify frame
297    for(int i=0; i<nframes; i++)
298      if(( framex-extend <= wmx && wmx <= framew+extend )
299          &&
300         ( frameh-((i+.5)*(frameh-framey)/(nframes-1)) <= wmy &&
301          wmy <= frameh-((i-.5)*(frameh-framey)/(nframes-1)) ))
302        {
303          *framepos = i;
304          return;
305        }
306
307    *framepos = active_frame;
308  }
309
310
311  // identifyrow - see if row number was changed
312  void
313  Left :: identifyrow(double wmx, double wmy, int *rowpos)
314  {
315    *rowpos = active_nrows;
316    // identify row
317    for(int i=0; i<nss; i++)
318      if(( framex-extend <= wmx && wmx <= framew+extend )
319          &&
320         ( rowh-((i+.5)*(rowh-rowy)/(nss-1)) <= wmy &&
321          wmy <= rowh-((i-.5)*(rowh-rowy)/(nss-1)) ))
322        {
323          *rowpos = i+1;
324          return;
325        }
326  }
327
 28
329  //identifytoggle - see if 'module lines' toggle has been changed
330  int
331  Left :: identifytoggle(double wmx, double wmy, Toggle *t )
332  {
333          if( (t->x-.4 <= wmx && wmx <= t->x+.4 ) &&
334                  ( t->y-.015 <= wmy && wmy <= t->y+.015 )) {
335                  *t->flag = !*t->flag;
336                  return TOGGLE;
```

```
337           }
338           return 0;
339   }
340
341
342   // drawframeslider - draw frame slider
343   void
344   Left :: drawframeslider(int frame)
345   {
346     color(MY_CYAN);
347     rectf(framex, framey, framew, frameh);
348     LineWidth(3);
349     active_frame = frame;
350
351     color(DK_YELLOW);
352     DrawLine(framex-extend, frameh-(frame*(frameh-framey)/(nframes-1)),
353                   framew+extend, frameh-(frame*(frameh-framey)/(nframes-1)));
354     DrawString(framew+2*extend, frameh-(frame*(frameh-framey)/(nframes-1))-
355                   .3*char_h, framenames[frame], VxDrawLeft);
356     color(MY_WHITE);
357     DrawString((framew-framex)/2+framex, frameh+.45*char_h,"FRAMES",VxDrawCenter);
358     DrawString((framew-framex)/2+framex, framey-char_h, "", VxDrawCenter);
359     LineWidth(1);
360   }
361
362
363   // drawrowslider - draws the row slider
364   void
365   Left :: drawrowslider(int nrows)
366   {
367     char titleside[MAX_STRING];
368
369     color(MY_CYAN);
370     rectf(framex, rowy, framew, rowh);
371     LineWidth(3);
372     color(DK_YELLOW);
373     double rowheight = rowh-((nrows-1)*(rowh-rowy)/(nss-1));
374     DrawLine(framex-extend, rowheight, framew+extend, rowheight);
375     sprintf(titleside, "%d", nrows);
376     color(MY_WHITE);
377     DrawString((framew-framex)/2+framex, rowh+1.3*char_h,"ROWS",VxDrawCenter);
378     DrawString((framew-framex)/2+framex, rowh+.3*char_h,"min",VxDrawCenter);
379     DrawString((framew-framex)/2+framex, rowy-.6*char_h,"max",VxDrawCenter);
380     LineWidth(1);
381   }
382
383
384   // drawtoggle - draw all of the toggles
385   void
386   Left :: drawtoggle(Toggle *t)
387   {
388     color( t->flag ? DK_YELLOW : DK_GREEN );
389     DrawString( t->x, t->y,
390                 (const char *)(t->labels[ *(t->flag) ]), VxDrawCenter);
391   }
392
```

```
393
394   void
395   Left :: timeout(Widget w)
396   {
397     Left *that = VxMotifCast( Left, w );
398
399     that->interval_id = XtAppAddTimeOut(
400           app_context,
401           frame_delay*10,
402           (XtTimerCallbackProc) Left :: timeout,
403           (XtPointer) w);
404
405     // switch to double time if redrawing is too slow
406     //if( redraw_time > frame_delay )
407     //  active_frame++;
408
409     active_frame = ++active_frame % nframes;
410     set_ssvar(active_ssvar, active_nrows);
411
412     pub.Changed(FRAME_CHANGE);
413   }
414
415
416   // this is a kluge
417   static GLXconfig glxConfig [] = {
418       { GLX_NORMAL, GLX_DOUBLE,   TRUE },
419       { GLX_NORMAL, GLX_COLORMAP, TRUE },
420       { GLX_NORMAL, GLX_ZSIZE,    GLX_NOCONFIG },
421       { 0, 0, 0 }
422   };
423
424
425   Left :: Left(char const *name, Widget parent ) :
426
427           // default constructors
428           VxGLDrawingArea(name, parent, glxConfig),
429           VxMouseable( VxDown | VxUp | VxDrag ),
430           VxFunctionality(this)
431   {
432           int i=0;
433           double where=1.01, delta=.03;
434
435           // set up my toggles
436           if( do_modules )
437                   toggles[i++] = new Toggle(
438                           .5, (where-=delta), &do_modline,
439                           "No Mod Lines", "Mod Lines", MODLINES_CHANGE );
 40
441           if( do_modules )
442                   toggles[i++] = new Toggle(
443                           .5, (where-=delta), &do_modnames,
444                           "No Mod Names", "Mod Names", MODNAMES_CHANGE );
445
446
447           toggles[i++] = new Toggle(
448                   .5, (where-=delta), &do_colors, "No Colors", "Colors", COLOR_CHANG
```

```
449
450           toggles[i++] = new Toggle(
451                   .5, (where-=delta), &do_showfill,
452                   "No Fill", "Fill", SHOWFILL_CHANGE );
453
454           toggles[i++] = new Toggle(
455                   .5, (where-=delta), &do_zoom,
456                   "Zoom", "Zoom Pending", ZOOM_PENDING );
457
458           if( nframes>1 )
459             toggles[i++] = new Toggle(
460                    .5, (where-=delta), &do_animate,
461                   "play", "stop", ANIMATE_CHANGE );
462
463           assert( i<MAX_TOGGLES );
464           toggles[i] = NULL;       // unnecessary!
465
466
467           // make my sliders
468           i = 0;
469           if( nframes>1 )
470               sliders[ i++ ] = new Slider(
471                   .15, .65, .15, .1,
472                   0, 100, &frame_delay,
473                   SPEED_CHANGE, "SPEED", "Fast", "Slow");
474
475           assert( i<MAX_SLIDERS );
476           sliders[i] = NULL;       // unnecessary!
477
478           // events that I care about
479           pub.Subscribe(this, ACTIVE_SSVAR);
480           pub.Subscribe(this, ACTIVE_MODVAR);
481           pub.Subscribe(this, ACTIVE_SS);
482           pub.Subscribe(this, ACTIVE_MODULE);
483           pub.Subscribe(this, ACTIVE_FILE);
484           pub.Subscribe(this, ANIMATE_CHANGE);
485           pub.Subscribe(this, COLOR_CHANGE);
486           pub.Subscribe(this, FRAME_CHANGE);
487           pub.Subscribe(this, MODLINES_CHANGE);
488           pub.Subscribe(this, MODNAMES_CHANGE);
489           pub.Subscribe(this, NROW_CHANGE);
490           pub.Subscribe(this, SHOWFILL_CHANGE);
491           pub.Subscribe(this, SPEED_CHANGE);
492           pub.Subscribe(this, ZOOM_PENDING);
493   }
494
495   void
 96   Left :: DoExpose(int, int, int, int)
497   {
498     GLXwinset( MyDisplay(), MyWindow());
499     ortho2(0, 1, 0, 1);
500
501     color(MY_BLACK);
502     clear();
503
504     // draw module statistics
```

```
505     if (do_modules) {
506       drawmodstats();
507       set_ssstats();
508     }
509
510     // draw subsystem statistics
511     drawssstats();
512
513     // draw frame slider
514     if (nframes > 1)
515           drawframeslider(active_frame);
516
517     // draw row slider
518     drawrowslider(active_nrows);
519
520     // draw toggles
521     for(int i=0; toggles[i]; i++)
522       drawtoggle(toggles[i]);
523
524     // draw the sliders
525     for(i=0; sliders[i]; i++)
526       drawslider(sliders[i]);
527
528     swapbuffers();
529   }
530
531
532   void
533   Left :: DoMouse(VxMouseActions what, int /*button*/,
534                         XEvent const * /*xevent*/)
535   {
536           double wmx, wmy;
537           int position, i;
538
539           switch (what) {
540
541           case VxDown:
542
543                   // convert mouse position to world coordinates
544                   wmx = XStoU( MouseX() );
545                   wmy = YStoU( Height()-1 - MouseY() );
546
547                   // check to see if active variable has been changed
548                   if (do_modules) {
549                           identifymodstat( wmx, wmy, &position );
550                           if ( position != ::active_modvar )
 51                          {
 52                                   ::active_modvar = position;
553                                  ::active_ssvar = position;
554                                  set_modvar(::active_modvar, active_nrows);
555                                  pub.Changed(ACTIVE_MODVAR);
556                           }
557                   }
558                   else {
959                           identifyssstat( wmx, wmy, &position );
 60                         if( position != ::active_ssvar )
```

```
561                             {
562                                     ::active_ssvar = position;
563                                     set_ssvar(::active_ssvar, active_nrows);
564                                     pub.Changed(ACTIVE_SSVAR);
565                             }
566                     }
567
568                     // check to see if active frame has been changed
569                     if (nframes > 1) {
570                             identifyframe( wmx, wmy, &position );
571                             if( position != ::active_frame )
572                             {
573                                     ::active_frame = position;
574                                     set_ssvar(active_ssvar, active_nrows);
575                                     pub.Changed(FRAME_CHANGE);
576                             }
577                     }
578
579                     // check to see if number of rows has been changed
580                     identifyrow( wmx, wmy, &position );
581                     if( position != ::active_nrows )
582                     {
583                             ::active_nrows = position;
584                             if (do_modules) {
585                                     modulelayout(active_nrows);
586                                     set_sslayout();
587                             }
588                             else
589                                     set_ssvar(active_ssvar, active_nrows);
590                             pub.Changed(NROW_CHANGE);
591                     }
592
593                     // check to see if module line toggle has been changed
594                     for(i=0; toggles[i]; i++)
595                             if( identifytoggle( wmx, wmy, toggles[i] ) )
596                               {
597                                 pub.Changed( toggles[i]->mypub );
598                                 break;
599                               }
600
601                     // check the sliders
602                     for(i=0; sliders[i]; i++)
603                             if( identifyslider( wmx, wmy, sliders[i] ) )
604                               {
605                                 pub.Changed( sliders[i]->mypub );
606                                 break;
607                               }
608
609                     break;
610
611             case VxDrag:
612
613                     // convert mouse position to world coordinates
614                     wmx = XStoU( MouseX() );
615                     wmy = YStoU( Height()-1 - MouseY() );
616
```

```
617                     // check to see if active frame has been changed
618                     if (nframes > 1) {
619                             identifyframe( wmx, wmy, &position );
620                             if( position != ::active_frame )
621                             {
622                                     ::active_frame = position;
623                                     set_ssvar(active_ssvar, active_nrows);
624                                     pub.Changed(FRAME_CHANGE);
625                             }
626                     }
627
628                     // check to see if number of rows has been changed
629                     identifyrow( wmx, wmy, &position );
630                     if( position != ::active_nrows )
631                     {
632                             ::active_nrows = position;
633                             if (do_modules) {
634                                     modulelayout(active_nrows);
635                                     set_sslayout();
636                             }
637                             else
638                                     set_ssvar(active_ssvar, active_nrows);
639                             pub.Changed(NROW_CHANGE);
640                     }
641
642                     // check the sliders
643                     for(i=0; sliders[i]; i++)
644                             if( identifyslider( wmx, wmy, sliders[i] ) )
645                               {
646                                 pub.Changed( sliders[i]->mypub );
647                                 break;
648                               }
649
650                 }
651     }
652
653     void
654     Left :: Update(int article,  int /* part */,  void *)
655     {
656
657             switch (article) {
658             case ANIMATE_CHANGE:
659               if( do_animate )
660                 {
661                   // turn on the animation
662                   interval_id = XtAppAddTimeOut(
663                             app_context,
 64                            frame_delay,
665                             (XtTimerCallbackProc) Left :: timeout,
666                             (XtPointer) widget);
667
668                 }
669               else
670                   // turn off the animation
671                   XtRemoveTimeOut(interval_id);
672
```

```
673             ExposeMe();
674             break;
675
676           case ACTIVE_SS:
677           case ACTIVE_MODULE:
678           case ACTIVE_SSVAR:
679           case ACTIVE_MODVAR:
680           case COLOR_CHANGE:
681           case FRAME_CHANGE:
682           case MODLINES_CHANGE:
683           case MODNAMES_CHANGE:
684           case NROW_CHANGE:
685           case SHOWFILL_CHANGE:
686           case SPEED_CHANGE:
687           case ZOOM_PENDING:
688             ExposeMe();
689             break;
690
691           default:
692             cout << __FILE__ << " " << __LINE__ << "ERROR - not reached"
693               << endl;
694           }
695   }
```

main.c

```
1   /*
2    * driver for seesys
3    */
4
5   #include <assert.h>
6   #if defined(SVR3)
7   #       include <getopt.h>
8   #endif
9   #include <iostream.h>
10  #include <search.h>
11  #include <stdlib.h>
12  #include <stdio.h>
13  #include <string.h>
14
15  #include "seesys.h"
16
17  // some extra manofest constants
18  #define MAX_SSVARS      15      /* max number of ss variables */
19  #define MAX_MODVARS     15      /* max number of module variables */
20  #define MAX_FRAMES      500
21
22  #ifndef TRUE
23  #       define TRUE     1
24  #       define FALSE    0
25  #endif
26
27  // default files
28  static char *ssnames_file       = "ssnames";
29  static char *modnames_file      = "modnames";
30  static char *framenames_file    = "framenames";
31
32  // default variables
33  static char *ssnames_main[MAX_SS];
34  static char *modnames_main[MAX_MOD];
35
36
37  // real memory for variables
38  static char   *ssvarnames_main[MAX_SSVARS];
39  static char   *ssfillnames_main[MAX_SSVARS];
40  static double ssvar_main[MAX_SSVARS*MAX_FRAMES];
41  static double ssfill_main[MAX_SSVARS*MAX_FRAMES];
42  static char   *modvarnames_main[MAX_MODVARS];
43  static char   *modfillnames_main[MAX_MODVARS];
44  static double modvar_main[MAX_MODVARS*MAX_FRAMES];
45  static double modfill_main[MAX_MODVARS*MAX_FRAMES];
46  static char   *framenames_main[MAX_FRAMES];
47  static int    flags_main[25];
48
49
50  static void
51  usage(char *progname)
52  {
53    cout << progname
54         << " [-f framenames_file]"
55         << " [-n] (show module names)"
56         << " [-m modvar modvar modvar ...]"
```

```
57          << " [-s ssnames_file]"
58          << " ssvar ssvar ssvar ..."
59          << endl;
60      exit(1);
61  }
62
63  void
64  read_ssvars(int argc, char **argv)
65  {
66          int i,j,k;
67          FILE *fh;
68          char *filename;
69          int inputnum[MAX_SSVARS];
70
71          for(i=0; optind < argc; optind++, nssvars++, nssfills++)
72          {
73                  filename = argv[optind];
74                  ssvarnames_main[nssvars] = strdup(filename);
75
76                  if( !(fh=fopen(filename, "r")) )
77                  {
78                          cerr << filename << "\tcannot open - quitting\n";
79                          exit(1);
80                  }
81
82                  int howm = 0;
83                  while( 1==fscanf(fh, "%lf", &ssvar_main[i] ) )
84                          i++, howm++;
85
86                  if( howm % nss )
87                  {
88                          cout << "weird - odd number of subsystem variables "
89                                          << "quitting" << endl;
90                          exit(1);
91                  }
92
93                  inputnum[nssvars] = howm;
94                  nframes = howm/nss;
95                  assert(nframes<MAX_FRAMES);
96
97                  fclose(fh);
98          }
99
100         // now read the corresponding fill statistics
101         for (i=0, j=0; i<nssfills; i++) {
102
103                 char filename[MAX_STRING];
104                 sprintf(filename, "%s.fil", ssvarnames_main[i]);
105                 ssfillnames_main[i] = ssvarnames_main[i];
106                 if( !(fh=fopen(filename, "r")) ) {
107                         do_fill[i] = 0;
108                         for (k=0; k<inputnum[i]; k++, j++)
109                                 ssfill_main[j] =0;
110                 }
111
112                 else {
```

```
113                             do_fill[i] = 1;
114                             int howm2 = 0;
115                             while( 1==fscanf(fh, "%lf", &ssfill_main[j] ) )
116                                     j++, howm2++;
117
118                             if( howm2 != inputnum[i] )
119                             {
120                                     cout << "weird - number of subsystem and "
121                                             << "fill variables do not match"
122                                            << "quitting" << endl;
123                                     exit(1);
124                             }
125                     }
126
127                     fclose(fh);
128             }
129     }
130
131
132     void
133     read_modvars(int argc, char **argv)
134     {
135             int i,j,k;
136             FILE *fh;
137             char *filename;
138             int inputnum[MAX_MODVARS];
139
140             for(i=0; optind < argc; optind++, nmodvars++, nmodfills++)
141             {
142                     filename = argv[optind];
143                     modvarnames_main[nmodvars] = strdup(filename);
144
145                     if( !(fh=fopen(filename, "r")) )
146                     {
147                             cerr << filename << "\tcannot open - quitting\n";
148                             exit(1);
149                     }
150
151                     int howm = 0;
152                     while( 1==fscanf(fh, "%lf", &modvar_main[i] ) )
153                             i++, howm++;
154
155                     if( howm%nmod )
156                     {
157                             cout << "weird - odd number of module variables "
158                                             << "quitting" << endl;
159                             exit(1);
 60                     }
161
162                     inputnum[nmodvars] = howm;
163                     nframes = howm/nmod;
164                     assert(nframes<MAX_FRAMES);
165
166                     fclose(fh);
167
168
```

```
169           // now read the corresponding fill statistics
170           for (i=0, j=0; i<nmodfills; i++) {
171
172                   char filename[MAX_STRING];
 73                   sprintf(filename, "%s.fil", modvarnames_main[i]);
174                   modfillnames_main[i] = modvarnames_main[i];
175                   if( !(fh=fopen(filename, "r")) ) {
176                           domod_fill[i] = 0;
177                           for (k=0; k<inputnum[i]; k++, j++)
 78                                   modfill_main[j] =0;
179                   }
180
181                   else {
182                           domod_fill[i] = 1;
183                           int howm2 = 0;
184                           while( 1==fscanf(fh, "%lf", &modfill_main[j] ) )
185                                   j++, howm2++;
186
187                           if( howm2 != inputnum[i] )
 88                           {
189                                   cout << "weird - number of module and "
190                                           << "fill variables do not match"
191                                          << "quitting" << endl;
192                                   exit(1);
193                           }
194                   }
195
196                   fclose(fh);
197           }
198   }
199
200
201   void
202   main(int argc, char **argv)
203   {
204     int c, i, n, errflg;
205     FILE *fh;
206     char buf[MAX_STRING];
207     int sscount = 0;
208
209     // kluge for now for debugging - core dump on abort
210     atexit(abort);
211
212     // handle the command line arguments
213     errflg = 0;
214     while ((c = getopt(argc, argv, "af:mns:")) != -1)
215       switch(c) {
 16       case 'a':
217         // set ss animation flag
218         do_animatelayout = 1;
219         break;
220
221       case 'f':
222         framenames_file = optarg;
223         break;
224
```

```
225       case 'n':
226         do_modnames = 1;
227         break;
228
229       case 's':
230         ssnames_file = optarg;
231         break;
232
233       case 'm': {
234          char savess[MAX_STRING];
235          char *subsystem, *module;
236
237          // set module flag
238         do_modules = 1;
239
240         // read in the modnames
241         if( !(fh=fopen( modnames_file, "r")) )
242               {
243                 cout << modnames_file << "\tcannot open - quitting\n";
244                 exit(1);
245               }
246
247         // read each line in file
248         for( nmod=0; fgets(buf, sizeof(buf), fh); nmod++)
249           {
250             if( nmod>=MAX_MOD )
251               {
252                 cout << "Too many modules (increase MAX_MOD) - quitting" << endl;
253                 exit(1);
254               }
255
256             if( (n=strlen(buf)) >= sizeof(buf) )
257               {
258                 cerr << "line " << nmod+1 << " too long in file "
259                   << ssnames_file << " quitting" << endl;
260                 exit(1);
261               }
262             buf[n-1] = '\0';  // zap newline
263
264             subsystem = strtok(buf, "/ \t");
265             module = strtok(NULL, "/ \t");
266
267             // first time in loop save the module count
268             if (nmod == 0)
269               strcpy(savess, buf);
270
271             if (strcmp(buf, savess) != 0) {
272               sscount++;
273               strcpy(savess, buf);
274             }
275
276             modulesPerSS[sscount]++;
277             modnames_main[nmod] = strdup( module );
278           }
279         fclose(fh);
280         break;
```

```
281       }
282
283       case '?':
284         errflg++;
285       }
286
287     if(errflg || optind>=argc)
288       usage(argv[0]);
289
290
291     // read in the ssnames
292     if( !(fh=fopen( ssnames_file, "r")) )
293           {
294             cerr << ssnames_file << "\tcannot open - quitting\n";
295             exit(1);
296           }
297     for( nss=0; fgets(buf, sizeof(buf), fh); nss++)
298       {
299         if( nss>=MAX_SS )
300           {
301             cout << "Too many subsystems - quitting" << endl;
302             exit(1);
303           }
304         n = strlen(buf);
305         if( n>=sizeof(buf) )
306           {
307             cerr << "line " << nss+1 << " too long in file "
308               << ssnames_file << " quitting" << endl;
309             exit(1);
310           }
311         buf[n-1] = '\0';  // zap newline
312         ssnames_main[nss] = strdup( buf );
313       }
314     fclose(fh);
315
316     if ((do_modules) && ( nss!= ++sscount ))
317     {
318       cout << "Module data does not exist for all subsystems - quitting" << endl;
319       exit(1);
320     }
321
322
323     if (do_modules) //read in the module variables
324           read_modvars(argc, argv);
325     else          // read in the subsystem variables
326           read_ssvars(argc, argv);
327
 28
329     if ((nframes > 1) && (fh=fopen(framenames_file, "r"))) {
330           i = 0;
331           while(fgets(buf, sizeof(buf), fh))
332                   framenames_main[i++] = strdup(buf);
333           fclose(fh);
334           if (i!=nframes) {
335                   cerr << "error - wrong number of frames" << endl;
336                   exit(1);
```

```
337         }
338     }
339     else
340         // make some default frame names
341         for(i=0; i<nframes; i++)
342         {
343                 sprintf(buf, "%d", i+1);
344                 framenames_main[i] = strdup( buf);
345         }
346
347     // default values of flags
348     flags_main[0] = FALSE;          // do_modline
349     flags_main[1] = do_modules;     // do_modules
350     flags_main[2] = FALSE;          // do_fork
351     flags_main[3] = TRUE;           // do_colors
352     flags_main[4] = do_animatelayout;      // ssanimate
353     flags_main[5] = TRUE;           // do_showfill
354     flags_main[6] = FALSE;          // do_zoom
355     flags_main[7] = do_modnames;    // print module names in zoom view
356
357
358
359     // do it!
360     seesys(
361             &argc,
362             argv,
363             ssnames_main,           // ss names
364             &nss,                   // number of subsystems
365             modnames_main,          // module names
366             &nmod,                  // number of modules
367
368             // subsystem variables
369             ssvar_main,             // ss variables
370             &nssvars,               // number of ss variables
371             ssvarnames_main,        // ss variable names
372             ssfill_main,            // ss fill variables
373             &nssfills,              // number of ss fill variables
374             ssfillnames_main,       // ss fill variable names
375
376             // module variables
377             modvar_main,            // module variables
378             &nmodvars,              // number of module variables
379             modvarnames_main,       // module variable names
380             modfill_main,           // module fill variables
381             &nmodfills,             // number of module fill variables
382             modfillnames_main,      // module fill variable names
383
384             // animation frames
385             framenames_main,        // frame names
386             &nframes,               // number of frames
387
388             flags_main
389
390
```

seesys.c

```
1   #include <assert.h>
2   #include <iostream.h>
3   #include <math.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <strings.h>
7   #include <sys/types.h>  /* for fork() */
8   #include <unistd.h>     /* for fork() */
9
10  #include <X11/Intrinsic.h>
11  #include <X11/StringDefs.h>
12  #include <Xm/MainW.h>
13  #include <Xm/RowColumn.h>
14  #include <Xm/PushB.h>
15  #include <Xm/CascadeB.h>
16  #include <Xm/Frame.h>
17  #include <Xm/Scale.h>
18  #include <Xm/Label.h>
19  #include <Xm/Form.h>
20  #include <X11/Xirisw/GlxMDraw.h>
21
22  #include <gl/gl.h>
23
24  // my includes
25  #include "bottom.h"
26  #include "left.h"
27  #include "sysview.h"
28  #include "seesys.h"
29  #include "zoomview.h"
30
31
32  // subsystem variables
33  char   **ssnames;                // subsystem names
34  int    nss;                      // number of subsystems
35  double minssvar;                 // smallest subsystem variable
36  double maxssvar;                 // largest subsystem variable
37  double maxssframevar;            // largest subsystem variable across all frames
38  double maxmodframevar;           // largest module variable across all frames
39  int    modulesPerSS[MAX_SS];     // number modules per subsystem
40  int    do_fill[MAX_SS];          // do fill stats exist
41
42  // slider colors
43  int    cols[PARTITIONS] = {
44          DK_BLUE,
45          DK_GREEN,
46          DK_YELLOW,
47          DK_RED,
48  };
49
50  // module variables
51  char   **modnames;               // module names
52  int    nmod;                     // number of modules
53  double maxmodvar;                // largest module variable
54  int    dòmod_fill[MAX_MOD];      // do fill stats exist
55
56
```

```
57   // multiple variables
58   double *ssvar_all;       // all subsystem statistics
59   char   **ssvarnames;     // subsystem variable names
60   int    nssvars;          // number of subsystem variables
61   double *ssfill_all;      // all subsystem fill statistics
62   char   **ssfillnames;    // subsystem fill names
63   int    nssfills;         // number of fill variables
64   double *modvar_all;      // all module statistics
65   char   **modvarnames;    // module variable names
66   int    nmodvars;         // number of module variables
67   double *modfill_all;     // all module fill statistics
68   char   **modfillnames;   // module fill names
69   int    nmodfills;        // number of fill variables
70
71
72   // animation variables
73   char   **framenames;     // frame names
74   int    nframes;          // how many frames
75   int    frame_delay;      // milliseconds to delay btw frames
76   int    redraw_time;      // redraw time in milliseconds
77
78   // zoom variables
79   ZOOM  zoom[MAX_ZOOM];
80
81   // state variables
82   int do_animatelayout;    // animate over frames
83   int do_animate;          // animate over frames
84   int do_colors;           // colors for ss or modules
85   int do_fork;             // should we fork?
86   int do_modline;          // lines between the modules
87   int do_modules;          // show the modules
88   int do_modnames;         // show module names in zoomview
89   int do_showfill;         // show fill statistic
90   int do_zoom;             // zoom pending
91
92
93   // active variables
94   double *ssvar;           // current subsystem variable
95   double *ssfill;          // current ss fill variable
96   double *modvar;          // current module variable
97   double *modfill;         // current module fill variable
98   int    active_ssvar;     // which ss var is active
99   int    active_modvar;    // which module var is active
100  int    active_frame;     // which frame is active
101  int    active_ss;        // which subsystem is mouse on
102  int    active_module;    // which module is mouse on
103  int    active_nrows;     // how many rows in display
104
105  // subsystem layout variables
106  double ssxpos[MAX_SS];
107  double ssypos[MAX_SS];
108  double sswidth[MAX_SS];
109  double ssheight[MAX_SS];
110  double ssmax[MAX_SS];
111  double ssaverage[MAX_SS];
112
```

```
113   // module layout variables
114   double modxpos[MAX_MOD];
115   double modypos[MAX_MOD];
116   double modwidth[MAX_MOD];
117   double modheight[MAX_MOD];
118
119
120   // publications
121   Publication pub;
122
123   // X junk
124   XtAppContext app_context;
125   Widget toplevel;
126
127   static String fallback_resources[] = {
128       "seesys.mainw.width: 800",
129       "seesys.mainw.height: 800",
130       "seesys*left.height: 800",
131       "seesys*left.width: 150",
132       "seesys*sysview.height: 650",
133       "seesys*sysview.width:  700",
134       "seesys*bottom.height: 100",
135       "seesys*bottom.width: 650",
136       "*frame*shadowType: SHADOW_IN",
137       NULL
138   };
139
140
141
142
143   // static function prototypes
144
145   static void set_ssframe(int whichframe);
146   void set_ssvar(int which, int nrows);
147   void set_modvar(int which, int nrows);
148   void set_sslayout();
149   void layout(int nrows);
150   void modulelayout(int nrows);
151   void ssanimate_layout(int nrows);
152   void set_maxssvar();
153   void set_maxssframevar();
154   void set_maxmodvar();
155   void set_maxmodframevar();
156   int  set_sscolor(int ss);
157   int  set_modcolor(int mod);
158
159
160
161
162   extern "C" void          // no name mangling
163   seesys(
164           int     *argc_ptr,
165           char    *argv[],
166           char    **ssnames_a,      // ss names
167           int     *nss_a,           // number of subsystems
168           char    **modnames_a,     // module names
```

```
169           int    *nmod_a,          // number of modules
170
171           // subsystem variables
172           double *ssvar_a,         // ss variables
173           int    *nssvars_a,       // number of ss variables
174           char   **ssvarnames_a,   // ss variable names
175           double *ssfill_a,        // ss fill variables
176           int    *nssfills_a,      // number of ss fill variables
177           char   **ssfillnames_a, // ss fill variable names
178
179           // module variables
180           double *modvar_a,        // module variables
181           int    *nmodvars_a,      // number of module variables
182           char   **modvarnames_a, // module variable names
183           double *modfill_a,       // module fill variables
184           int    *nmodfills_a,     // number of module fill variables
185           char   **modfillnames_a,// module fill variable names
186
187           // animation frames
188           char   **framenames_a,  // frame names
189           int    *nframes_a,       // number of frames
190
191           int    *flags
192   ) {
193           /* save variables in statics */
194           ssnames  = ssnames_a;
195           nss      = *nss_a;
196           modnames  = modnames_a;
197           nmod      = *nmod_a;
198
199           // subsystem variables
200           ssvar_all = ssvar_a;
201           nssvars = *nssvars_a;
202           ssvarnames = ssvarnames_a;
203           ssfill_all = ssfill_a;
204           nssfills = *nssfills_a;
205           ssfillnames = ssfillnames_a;
206
207           // module variables
208           modvar_all = modvar_a;
209           nmodvars = *nmodvars_a;
210           modvarnames = modvarnames_a;
211           modfill_all = modfill_a;
212           nmodfills = *nmodfills_a;
213           modfillnames = modfillnames_a;
214
215           // animation frames
216           framenames = framenames_a;
217           nframes = *nframes_a;
218
219           // flags
220           do_modline      = flags[0];
221           do_modules      = flags[1];
222           do_fork         = flags[2];
223           do_colors       = flags[3];
224           do_animatelayout= flags[4];
```

```
225           do_showfill     = flags[5];
226           do_zoom         = flags[6];
227           do_modnames     = flags[7];
228
229
230           // initialization
231           do_animate = FALSE;
232           active_frame = 0;
233           active_nrows = (int)(ceil( sqrt( (double)nss ) ));
234           active_ss = -1;
235           frame_delay = 10;       // milliseconds
236
237           if (do_modules)
238                   set_modvar(0, active_nrows);
239           else
240                   set_ssvar(0, active_nrows);
241
242
243           // should we fork?
244           int pid = do_fork ? fork() : 0;
245
246           if(pid < 0)
247             cerr << "Could not fork\n";
248           else if(pid > 0)
249             return;       /* parent exits immediately */
250
251           deflinestyle(DASH_LINE, 0xF0F0);
252
253
254           // now make the windows
255           Widget mainw, frame;
256           int n;
257           Arg args[20];
258
259
260           toplevel = XtAppInitialize(&app_context, "seesys",
261                                      (XrmOptionDescList)NULL , 0,
262   #if defined(SVR3) || defined(sun)
263           /* kluge for different version of X */
264           (Cardinal *)
265   #endif
266                                      argc_ptr,
267                                      (String*)argv,
268                                      fallback_resources,
269                                      (ArgList)NULL, 0);
270
271           // set the window title
272           XtVaSetValues( toplevel,
273                   XmNtitle, argv[0],
274                   NULL
275                 );
276
277
278
279       n = 0;
280       mainw = XtCreateManagedWidget("mainw",
```

```
281                 xmMainWindowWidgetClass, toplevel,
282                 args, n);
283
284     frame = XtVaCreateManagedWidget("frame",
285         xmFormWidgetClass, mainw,
286         NULL);
287
288
289     // controls along the left
290     Left left("left", frame);
291     XtVaSetValues( left.widget,
292         XmNleftAttachment,      XmATTACH_FORM,
293         XmNtopAttachment,       XmATTACH_FORM,
294         XmNbottomAttachment,    XmATTACH_FORM,
295         NULL
296     );
297
298
299      // bottomspace
300     Bottom bottom("bottom", frame);
301     XtVaSetValues( bottom.widget,
302         XmNbottomAttachment,    XmATTACH_FORM,
303         XmNleftAttachment,      XmATTACH_WIDGET,
304         XmNleftWidget,          left.widget,
305         XmNrightAttachment,     XmATTACH_FORM,
306         NULL
307     );
308
309
310     // system view
311     Sysview sysview("sysview", frame);
312     XtVaSetValues( sysview.widget,
313         XmNrightAttachment,     XmATTACH_FORM,
314         XmNtopAttachment,       XmATTACH_FORM,
315         XmNleftAttachment,      XmATTACH_WIDGET,
316         XmNleftWidget,          left.widget,
317         XmNbottomAttachment,    XmATTACH_WIDGET,
318         XmNbottomWidget,        bottom.widget,
319         NULL
320     );
321
322     short red[100], green[100], blue[100];
323     for(int i=0; i<16; i++)
324         getmcolor(i, &red[i], &green[i], &blue[i] );
325
326     mapcolor(MY_BLACK, 0, 0, 0);
327     mapcolor(MY_WHITE, 255, 255, 255);
328     mapcolor(MY_CYAN, 0, 255, 255);
329     mapcolor(MY_GRAY, 125, 125, 125);
330     mapcolor(DK_GRAY, 50, 50, 50);
331     mapcolor(LT_GRAY, 185, 185, 185);
332     mapcolor(DK_BLUE, 0, 0, 140);
333     mapcolor(LT_BLUE, 0, 0, 190);
334     mapcolor(DK_GREEN, 0, 145, 0);
335     mapcolor(LT_GREEN, 0, 180, 0);
336     mapcolor(DK_YELLOW, 255, 215, 0);
```

```
337       mapcolor(LT_YELLOW, 255, 255, 0);
338       mapcolor(DK_RED, 200, 0, 0);
339       mapcolor(LT_RED, 230, 0, 0);
340
341       // make the widgets
342       XtRealizeWidget(toplevel);
343
344       // ************** here we go *********************
345       XtAppMainLoop(app_context);
346
347       // put back old colors
348       for(i=0; i<16; i++)
349           mapcolor(i, red[i], green[i], blue[i] );
350
351       // we're done
352       if( pid>0 )
353         exit(0);      // child process exits
354   }
355
356
357   // update the active frame
358   static void
359   set_ssframe(int whichframe)
360   {
361           whichframe = whichframe % nframes;
362
363           if( whichframe!= active_frame )
364             {
365               active_frame = whichframe;
366               set_ssvar( active_ssvar, active_nrows );
367               pub.Changed(FRAME_CHANGE);
368             }
369   }
370
371
372
373   // change subsystem variables
374   void
375   set_ssvar(int which, int nrows)
376   {
377     active_ssvar = which;
378     ssvar = &ssvar_all[ active_ssvar * nframes * nss
379                 + active_frame * nss ];
380     ssfill = &ssfill_all[ active_ssvar * nframes * nss
381                 + active_frame * nss ];
382
383     set_maxssvar();
384     set_maxssframevar();
385     if (do_animatelayout)
386           ssanimate_layout(nrows);
387     else
388           layout(nrows);
389   }
390
391
392   // change module variables
```

```
393   void
394   set_modvar(int which, int nrows)
395   {
396     active_modvar = which;
397     modvar = &modvar_all[ active_modvar * nframes * nmod
398                + active_frame * nmod ];
399     modfill = &modfill_all[ active_modvar * nframes * nmod
400                + active_frame * nmod ];
401
402     set_maxmodvar();
403     set_maxmodframevar();
404     modulelayout(nrows);
405     set_sslayout();
406   }
407
408
409   // get_ssvar - returns statistic index for subsystems
410   double
411   get_ssvar(int whichvar, int whichss)
412   {
413     return ssvar_all[ whichvar * nframes * nss
414                       + active_frame * nss
415                       + whichss];
416   }
417
418
419   // get_modvar - returns statistic index for modules
420   double
421   get_modvar(int whichvar, int whichmod)
422   {
423     return modvar_all[ whichvar * nframes * nmod
424                       + active_frame * nmod
425                       + whichmod];
426   }
427
428
429   // layout - calculate ssxpos and ssypos
430   void
431   layout(int nrows)
432   {
433           int i, j, frame;
434           double sstot, maxsstot=0;
435
436           for(frame=0; frame<nframes; frame++)
437             {
438               for(i=0, sstot=0; i<nss; i++)
439                   sstot += ssvar_all[ active_ssvar*nframes*nss
440                             + frame*nss + i ];
441               maxsstot = MAX(sstot, maxsstot);
442             }
443
444           double perrow = maxsstot/nrows;
445
446           // position each row
447           int ss=0, howm;
448           double cumx, cumy=0;
```

```
449           while( ss<nss )
450             {
451               // how much goes on this row
452               double rowtot = 0;
453               for(howm=0, j=ss;
454                   rowtot<perrow && j<nss;
455                   j++, howm++)
456                 rowtot += ssvar[j];
457
458               cumy += rowtot;
459               for(i=0, cumx=0, j=ss; i<howm; j++, i++)
460                 {
461                   ssxpos[j] = cumx/rowtot;
462                   ssypos[j] = 1 - cumy/maxsstot;
463                   sswidth[j] = ssvar[j]/rowtot;
464                   ssheight[j] = rowtot/maxsstot;
465
466                   cumx += ssvar[j];
467                 }
468               ss += howm;
469             }
470   }
471
472
473
474   // modulelayout - calculate modxpos and modypos
475   void
476   modulelayout(int nrows)
477   {
478           int i, j, k, frame;
479           double modtot, maxmodtot=0;
480
481           for(frame=0; frame<nframes; frame++)
482             {
483               for(i=0, modtot=0; i<nmod; i++)
484                   modtot += modvar_all[ active_modvar*nframes*nmod
485                             + frame*nmod + i ];
486               maxmodtot = MAX(modtot, maxmodtot);
487             }
488
489           double perrow = maxmodtot/nrows;
490
491           // position each row
492           int mod=0, ss=0, howm, offset=0;
493           double cumx, cumy=0;
494
495           while( ss<nss )
496             {
497               // how much goes on this row
498               double rowtot = 0;
499               for(howm=0, j=offset;
500                   rowtot<perrow && j<nmod;
501                   j=offset, ss++) {
502                 for(k=0; k<modulesPerSS[ss]; k++)
503                   rowtot += modvar[j+k];
504                 howm += modulesPerSS[ss];
```

```
505                 offset += modulesPerSS[ss];
506               }
507
508               cumy += rowtot;
509               for(i=0, cumx=0, j=mod; i<howm; j++, i++)
510                 {
511                   modxpos[j] = cumx/rowtot;
512                   modypos[j] = 1 - cumy/maxmodtot;
513                   modwidth[j] = modvar[j]/rowtot;
514                   modheight[j] = rowtot/maxmodtot;
515
516                   cumx += modvar[j];
517                 }
518               mod += howm;
519             }
520   }
521
522
523   // ssanimate_layout - calculate ssxpos and ssypos
524   void
525   ssanimate_layout(int nrows)
526   {
527           int i, j, frame;
528           double maxsstot=0;
529
530           // do in initialization code?
531           for(i=0; i<nss; i++)
532             {
533               double m = 0;
534               for(frame=0; frame<nframes; frame++)
535                 m = MAX( m, ssvar_all[ active_ssvar*nframes*nss
536                                                 + frame*nss + i ] );
537               ssmax[i] = m;
538             }
539
540           for(i=0; i<nss; i++)
541                   maxsstot += ssmax[i];
542           double perrow = maxsstot/nrows;
543
544           // position each row
545           int ss=0, howm;
546           double cumx, cumy=0;
547           while( ss<nss )
548             {
549               // how much goes on this row
550               double rowtot = 0;
551               for(howm=0, j=ss;
552                   rowtot<perrow && j<nss;
553                   j++, howm++)
554                 rowtot += ssmax[j];
555
556               cumy += rowtot;
557               for(i=0, cumx=0, j=ss; i<howm; j++, i++)
558                 {
559                   ssxpos[j] = cumx/rowtot;
 60                  ssypos[j] = 1 - cumy/maxsstot;
```

```
561                     sswidth[j] = ssmax[j]/rowtot;
562                     ssheight[j] = rowtot/maxsstot;
563
564                     cumx += ssmax[j];
565                   }
566               ss += howm;
567             }
568   }
569
570
571   // set_sslayout - assigns ssxpos and ssypos values for module input
572   void
573   set_sslayout()
574   {
575           int offset = 0;
576
577           for (int i=0; i<nss; i++) {
578                   ssxpos[i] = modxpos[offset];
579                   ssypos[i] = modypos[offset];
580                   sswidth[i] = 0;
581                   for (int j=offset; j<offset+modulesPerSS[i]; j++)
582                           sswidth[i] += modwidth[j];
583                   //sswidth[i] = modxpos[offset+modulesPerSS[i]]-modxpos[offset];
584                   ssheight[i] = modheight[offset];
585                   offset += modulesPerSS[i];
586           }
587   }
588
589
590   // set_maxssvar - assigns largest subsystem variable to maxssvar
591   void
592   set_maxssvar()
593   {
594
595           maxssvar = ssvar[0];
596           for (int i=1; i < nss; i++) {
597                   if (ssvar[i] > maxssvar)
598                           maxssvar = ssvar[i];
599           }
600
601   }
602
603
604   // set_maxssframevar - assigns largest subsystem variable of
605   // all frames to maxssframevar.
606   void set_maxssframevar()
607   {
608           int frame, i, j, offset;
609
610           offset = active_ssvar * nframes * nss;
611           maxssframevar = ssvar_all[offset];
612           for (frame=0; frame<nframes; frame++)
613                   for (i=offset, j=0; j < nss; j++, i++, offset++)
614                           if (ssvar_all[offset] > maxssframevar)
615                                   maxssframevar = ssvar_all[offset];
616
```

```
617
618
619   // set_maxmodvar - assigns largest module variable to maxssvar
620   void
621   set_maxmodvar()
622   {
623           maxmodvar = modvar[0];
624           for (int i=1; i < nmod; i++) {
625                   if (modvar[i] > maxmodvar)
626                           maxmodvar = modvar[i];
627           }
628   }
629
630
631   // set_maxmodframevar - assigns largest module variable of
632   // all frames to maxmodframevar.
633   void set_maxmodframevar()
634   {
635           int frame, i, j, offset;
636
637           offset = active_modvar * nframes * nmod;
638           maxmodframevar = modvar_all[offset];
639           for (frame=0; frame<nframes; frame++)
640                   for (i=offset, j=0; j < nmod; j++, i++, offset++)
641                           if (modvar_all[offset] > maxmodframevar)
642                                   maxmodframevar = modvar_all[offset];
643   }
644
645
646   // set_sscolor - assigns a color to subsystem ss
647   int
648   set_sscolor(int ss)
649   {
650           int col;
651           double block;
652           if (nframes > 1)
653                   block = maxssframevar / PARTITIONS;
654           else
655                   block = maxssvar / PARTITIONS;
656           for (int i=1; i<=PARTITIONS; i++)
657                   if (ssvar[ss] <= i*block) {
658                           col = cols[i-1];
659                           break;
660                   }
661           return col;
662   }
663
664
665   // set_modcolor - assigns a color to module mod
666   int
667   set_modcolor(int mod)
668   {
669           int col;
670           double block;
671           if (nframes > 1)
672                   block = maxmodframevar / PARTITIONS;
```

```
673             else
674                     block = maxmodvar / PARTITIONS;
675             for (int i=1; i<=PARTITIONS; i++)
676                     if (modvar[mod] <= i*block) {
677                             col = cols[i-1];
678                             break;
679                     }
680             return col;
681     }
682
683     // - warning - this should be somewhere else
684     void
685     zoomview_quitCB(
686         Widget /*w*/,
687         caddr_t client_data,
688         caddr_t /*call_data*/
689     ) {
690
691       Zoomview *zoomview =  (Zoomview *)client_data; /*VxMotifCast( Zoomview, w );*/
692
693       int mybar = zoomview->mybar;
694
695       // mark these bars as unused
696       zoom[mybar].used = 0;
697
698       delete zoomview;
699     }
700
701
```

SYSVIEW.C

```
1   #include <assert.h>
2   #include <iostream.h>
3   #include <memory.h>
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <string.h>
7   #include <time.h>
8
9   #include <X11/Intrinsic.h>
10  #include <X11/StringDefs.h>
11  #include <Xm/MainW.h>
12  #include <Xm/RowColumn.h>
13  #include <Xm/PushB.h>
14  #include <Xm/CascadeB.h>
15  #include <Xm/Frame.h>
16  #include <Xm/Scale.h>
17  #include <Xm/Label.h>
18  #include <Xm/Form.h>
19  #include <X11/Xirisw/GlxMDraw.h>
20
21  #include <gl/gl.h>
22
23  #include "sysview.h"
24  #include "seesys.h"
25  #include "zoomview.h"
26
27
28  void
29  Sysview :: identify(double wmx, double wmy,
30                      int *ss, int *module, int *file)
31  {
32    *ss = *module = *file = -1;
33
34    // identify the active subsystem
35    for(int i=0; i<nss; i++)
36      if( ssxpos[i] <= wmx && wmx <= ssxpos[i]+sswidth[i]
37          &&
38         ssypos[i] <= wmy && wmy <= ssypos[i]+ssheight[i] )
39        {
40          *ss = i;
41          break;
42        }
43
44    // identify the active module
45    for(i=0; i<nmod; i++)
46      if( modxpos[i] <= wmx && wmx <= modxpos[i]+modwidth[i]
47          &&
48         modypos[i] <= wmy && wmy <= modypos[i]+modheight[i] )
49        {
50          *module = i;
51          break;
52        }
53
54    // now get the file
55  }
56
```

```
57
58  void
59  Sysview :: drawmod(
60          int i,              /* module to draw */
61          int col             /* it's color */
62  ) {
63        int j,k,ss;
64        double x1, x2, y1, y2, filly;
65        x1 = modxpos[i];
66        x2 = modxpos[i] + modwidth[i];
67
68        y1 = modypos[i];
69        y2 = modypos[i] + modheight[i];
70        filly = modfill[i];
71
72        // draw module
73        LineWidth(2);
74        color(do_colors ? col : DK_GRAY);
75        FillRect(x1, y1, x2-x1, y2-y1);
76        FrameRect(x1, y1, x2-x1, y2-y1);
77
78        // What subsystem is this module in?
79        for (ss=j=k=0; j<nss; j++)
80          if ((k += modulesPerSS[j]) <= i)
81            ss++;
82
83        // ss is my subysystem!
84
85        // draw subsystem fill average
86        color(LT_GRAY);
87        FillRect(modxpos[i], modypos[i], modwidth[i],
88          ssaverage[ss]*ssheight[ss]);
89        FrameRect(modxpos[i], modypos[i], modwidth[i],
90          ssaverage[ss]*ssheight[ss]);
91
92        // draw fill
93        if (do_showfill) {
94          color( do_colors ? MY_GRAY : LT_BLUE );
95          filly = modvar[i] ? modfill[i]/modvar[i] * modheight[i] : 0;
96          FillRect(x1, y1, x2-x1, filly);
97          FrameRect(x1, y1, x2-x1, filly);
98        }
99
100       // outline
101       if (do_modline) {
102         color(MY_WHITE);
103         FrameRect(x1, y1, x2-x1, y2-y1);
104       }
105
106       // outline active module
107       if (::active_module == i) {
108         color(DK_RED);
109         FrameRect(x1, y1, x2-x1, y2-y1);
110       }
111       LineWidth(1);
112 }
```

```
113
114
115   // drawssboundaries - draws ss outlines (when there is module input)
116   void
117   Sysview :: drawssboundaries(int ss)
118   {
119           double x1, x2, y1, y2;
120
121           x1 = ssxpos[ss];
122           x2 = ssxpos[ss] + sswidth[ss];
123           y1 = ssypos[ss];
124           y2 = ssypos[ss] + ssheight[ss];
125
126           // outline
127           LineWidth(4);
128           if (::active_ss == ss)
129                   color(DK_RED);
130           else
131                   color(MY_WHITE);
132           FrameRect(x1, y1, x2-x1, y2-y1);
133           LineWidth(1);
134
135           // ss name
136           color(MY_WHITE);
137           DrawString( .5*(x1+x2), .5*(y1+y2), ssnames[ss],
138                   VxDrawCenter );
139   }
140
141
142   void
143   Sysview :: drawss(int i, int col)
144   {
145         double x1, x2, y1, y2, filly, fillx;
146         x1 = ssxpos[i];
147         x2 = ssxpos[i] + sswidth[i];
148
149         y1 = ssypos[i];
150         y2 = ssypos[i] + ssheight[i];
151         filly = ssfill[i];
152
153         // draw subsystem
154         color(do_colors ? col : DK_GRAY );
155         FillRect(x1, y1, x2-x1, y2-y1);
156
157         // black fill for animate
158         if (do_animatelayout)
159           {
 60           color(MY_BLACK);
161           fillx = (1-ssvar[i]/(ssmax[i]+!ssmax[i]))* sswidth[i];
162           FillRect(x2-fillx, y1, fillx, y2-y1);
163           }
164         else
165           {
166             // draw fill
167               if (do_showfill) {
168                 color(MY_GRAY);
```

```
169                 filly = ssfill[i] (ssvar[i]+'ssvar[i]) * ssheight[i];
170                 FillRect(x1, y1, x2-x1, filly);
171               }
172           }
173
174         // outline
175         LineWidth(4);
176         color(::active_ss == i ? DK_RED : MY_WHITE);
177         FrameRect(x1, y1, x2-x1, y2-y1);
178         LineWidth(1);
179   }
180
181
182   // drawssname - draws subsystem i's name
183   void
184   Sysview :: drawssname(int ss)
185   {
186         double x1, x2, y1, y2;
187         x1 = ssxpos[ss];
188         x2 = ssxpos[ss] + sswidth[ss];
189
190         y1 = ssypos[ss];
191         y2 = ssypos[ss] + ssheight[ss];
192
193         // ss name
194         color(MY_WHITE);
195         DrawString( .5*(x1+x2), .5*(y1+y2), ssnames[ss],
196                     VxDrawCenter );
197   }
198
199
200   // calc_fillaverage - calculates the average ss fill
201   void
202   Sysview :: calc_fillaverage()
203   {
204           int i,j,k=0;
205           double moduletot, sstot;
206
207           for (i=0; i<nss; i++) {
208                   for (moduletot=0, sstot=0, j=0; j<modulesPerSS[i]; j++, k++) {
209                           moduletot += modfill[k];
210                           sstot += modvar[k];
211                   }
212                   ssaverage[i] = sstot ? moduletot/sstot : 0;
213           }
214   }
215
 16
217   // this is a kluge
218   static GLXconfig glxConfig [] = {
219       { GLX_NORMAL, GLX_DOUBLE,   TRUE },
220       { GLX_NORMAL, GLX_COLORMAP, TRUE },
221       { GLX_NORMAL, GLX_ZSIZE,    GLX_NOCONFIG },
222       { 0, 0, 0 }
223   };
224
```

```
225  Sysview :: Sysview(char const *name, Widget parent ) :
226
227          // default constructors
228          VxGLDrawingArea(name, parent, glxConfig),
229          VxMouseable( VxAll ),
230          VxFunctionality(this)
231  {
232          active_ss = -1;
233          active_module = -1;
234
235          // events that I care about
236          pub.Subscribe(this, ACTIVE_SS);
237          pub.Subscribe(this, ACTIVE_MODULE);
238          pub.Subscribe(this, ACTIVE_FILE);
239          pub.Subscribe(this, ACTIVE_SSVAR);
240          pub.Subscribe(this, ACTIVE_MODVAR);
241          pub.Subscribe(this, COLOR_CHANGE);
242          pub.Subscribe(this, FRAME_CHANGE);
243          pub.Subscribe(this, MODLINES_CHANGE);
244          pub.Subscribe(this, NROW_CHANGE);
245          pub.Subscribe(this, SHOWFILL_CHANGE);
246  }
247
248  void
249  Sysview :: DoExpose(int, int, int, int)
250  {
251    //struct timeval t1, t2;
252    //gettimeofday(&t1, NULL);
253
254    GLXwinset( MyDisplay(), MyWindow());
255    UserCoord(-0.01, 1.01,-0.01, 1.01);
256
257    // clear window
258    color(MY_BLACK);
259    Clear();
260
261    // draw animation outline
262    if (nframes > 1) {
263          LineWidth(3);
264          color(MY_WHITE);
265          FrameRect(0, 0, 1, 1);
266          LineWidth(1);
267    }
268
269    if (do_modules) {
270          calc_fillaverage();
271          for(int i=0; i<nmod; i++)
272                  drawmod(i, set_modcolor(i)+i%2);
273          for(int j=0; j<nss; j++)
274                  drawssboundaries(j);
275    }
276    else  // draw the subsystems
277          for(int i=0; i<nss; i++)
278                  drawss(i, set_sscolor(i));
279
280    // draw subsystem names
```

```
281     color(MY_WHITE);
282     for (int i=0; i<nss; i++)
283           drawssname(i);
284
285     swapbuffers();
286
287     //gettimeofday(&t2, NULL);
288     //redraw_time = (int)((t2.tv_sec-t1.tv_sec)*1000 +
289     //                    (t2.tv_usec-t1.tv_usec)/1000.);
290
291   }
292
293
294   void
295   Sysview :: DoMouse(VxMouseActions what, int button,
296                      XEvent const * /*xevent*/)
297   {
298     double wmx, wmy;
299     int ss, module, file;
300
301     // convert mouse position to world coordinates
302     wmx = XStoU( MouseX() );
303     wmy = YStoU( Height()-1 - MouseY() );
304
305     identify( wmx, wmy, &ss, &module, &file);
306
307     if( ss!=::active_ss )
308       {
309         ::active_ss = ss;
310         pub.Changed(ACTIVE_SS, ::active_ss);
311       }
312     if( module!=::active_module )
313       {
314         ::active_module = module;
315           pub.Changed(ACTIVE_MODULE, ::active_module );
316       }
317
318     switch( what ) {
319     case VxMotion:
320       break;
321
322     case VxDown:
323       if (button != 1)
324         break;
325
326       // get here if button 1
327       if (do_zoom)
328         {
329           Widget optd, frame;
330
331           // The shell for it to be in
332           optd = XtVaCreatePopupShell("Zoom",
333                   topLevelShellWidgetClass,
334                   toplevel,
335                   XmNdeleteResponse, XmDESTROY,
336                   XmNx, 200,
```

```
337                 XmNy, 500,
338                 XmNheight, 500,
339                 XmNwidth, 500,
340                 NULL);
341
342         frame = XtVaCreateManagedWidget("frame",
343                         xmFormWidgetClass, optd,
344                         NULL);
345
346         // find a free slot (start at 1!)
347         for(int i=1; i<MAX_ZOOM; i++)
348           if( !zoom[i].used )
349             break;
350
351         if( i == MAX_ZOOM )
352           {
353             // no free slots found
354             cout << "Error - No more zoom windows" << endl;
355             return;
 56           }
357
358         // window title is active subsystem
359         XtVaSetValues( optd,
360                 XmNtitle, ssnames[ ::active_ss ],
361                 NULL
362               );
363
364
365         zoom[i].ss = ::active_ss;
366         zoom[i].used = TRUE;
367
368         // create the zoom window
369         Zoomview *zoomview = new Zoomview( "subsys",
,70                             frame, i);
371
372         // attach widget to the frame
373         XtVaSetValues( zoomview->widget,
374                 XmNleftAttachment,      XmATTACH_FORM,
375                 XmNrightAttachment,     XmATTACH_FORM,
376                 XmNbottomAttachment,    XmATTACH_FORM,
377                 XmNtopAttachment,       XmATTACH_FORM,
378                 NULL
379               );
380
381         // what to do when it's killed
382         XtAddCallback( zoomview->widget, XtNdestroyCallback,
383                        (XtCallbackProc)zoomview_quitCB, (XtPointer)zoomview
 34                     );
385
386         // pop it up, don't grab any focus
387         XtPopup(optd, XtGrabNone);
388
389         // redraw overview window
390         pub.Changed(ZOOMSS_CHANGE, i);
391
392         do_zoom = 0;
```

```
393             pub.Changed(ZOOM_PENDING);
394           }
395         break;
396
397       case VxDrag:
398         break;
399       }
400     }
401
402     void
403     Sysview :: Update(int article,  int /* part */,  void *)
404     {
405       switch(article) {
406
407       case ACTIVE_SS:
408
409         GLXwinset( MyDisplay(), MyWindow());
410         frontbuffer(TRUE);
411
412         if (do_modules) {
413           // clear last one (unless it is the 1st active ss)
414           if (active_ss >= 0) {
415             drawssboundaries(active_ss);
416             last_ss = active_ss;
417           }
418
419           // reset variable
420           active_ss = ::active_ss;
421
422           // redraw new one
423           if (active_ss >= 0) {
424             color(DK_RED);
425             drawssboundaries(active_ss);
426           }
427         }
428         else {
429           // clear last one
430           if(active_ss >= 0 ) {
431             drawss( active_ss, set_sscolor(active_ss) );
432             drawssname( active_ss );
433           }
434
435           // reset variable
436           active_ss = ::active_ss;
437
438           // redraw new one
439           if(active_ss >= 0 ) {
440             drawss( active_ss, set_sscolor(active_ss) );
441             drawssname( active_ss );
442           }
443         }
444
445         frontbuffer(FALSE);
446         break;
447
448       case ACTIVE_MODULE:
```

```
449
450       GLXwinset( MyDisplay(), MyWindow());
451       frontbuffer(TRUE);
452
453       if (do_modules) {
454         // clear last one (unless it is the 1st active module)
455         if (active_module >= 0) {
456           drawmod(active_module, set_modcolor(active_module)+active_module%2);
457
458           // redraw ss boundaries and name
459           if (active_ss >= 0)
460             drawssboundaries(active_ss);
461
462           // redraw last ss
463           if (last_ss >= 0)
464             drawssboundaries(last_ss);
465         }
466
467         // reset variable
468         active_module = ::active_module;
469
470         // redraw new one
471         if (active_module >= 0) {
472           drawmod(active_module, set_modcolor(active_module)+active_module%2);
473
474           // redraw ss boundaries and name
475           if (active_ss >= 0)
476             drawssboundaries(active_ss);
477         }
478       }
479
480       frontbuffer(FALSE);
481       break;
482
483     case ACTIVE_SSVAR:
484     case ACTIVE_MODVAR:
485     case COLOR_CHANGE:
486     case FRAME_CHANGE:
487     case MODLINES_CHANGE:
488     case NROW_CHANGE:
489     case SHOWFILL_CHANGE:
490           ExposeMe();
491           break;
492
493     default:
494       cout << __FILE__ << __LINE__ << " Error - not reached" << endl;
495     }
 96     gflush();
497   }
```

zoomview.c

```
1   #include <assert.h>
2   #include <iostream.h>
3   #include <math.h>
4   #include <memory.h>
5   #include <stdio.h>
6
7   #include <Xm/DialogS.h>
8   #include <Xm/Form.h>
9   #include <X11/Xirisw/GlxMDraw.h>
10
11  #include "zoomview.h"
12  #include "seesys.h"
13
14
15  void
16  Zoomview :: identify(double wmx, double wmy,
17                       int *sschange, int *module, int *file)
18  {
19    int i, mod;
20    *sschange = *module = *file = -1;
21    int ss = zoom[mybar].ss;
22
23    // identify the active subsystem (always the zoom subsystem)
24    *sschange = ss;
25
26    // identify the active module
27    for(i=0, mod=min_mod; i<modulesPerSS[ss]; i++, mod++)
28      if( modx[i] <= wmx && wmx <= modx[i]+modw[i]
29          &&
30         0 <= wmy && wmy <= 1 )
31        {
32          *module = mod;
33          break;
34        }
35
36    // now get the file
37  }
38
39
40  void
41  Zoomview :: drawss(int i, int col)
42  {
43        double x1, x2, y1, y2, filly, fillx;
44        x1 = 0;
45        x2 = 1;
46
47        y1 = 0;
48        y2 = 1;
49
50        // draw subsystem
51        color(do_colors ? col : DK_GRAY );
52        FillRect(x1, y1, x2-x1, y2-y1);
53
54        // black fill for animate
55        if (do_animatelayout)
56
```

```
57              color(MY_BLACK);
58              fillx = (1-ssvar[i]/ssmax[i]+!ssmax[i]))* x2;
59              FillRect(x2-fillx, y1, fillx, y2);
60            }
61          else
62            {
63              // draw fill
64                if (do_showfill) {
65                  color(MY_GRAY);
66                  filly = ssfill[i]/(ssvar[i]+!ssvar[i]) * y2;
67                  FillRect(x1, y1, x2, filly);
68                }
69            }
70
71          // outline
72          LineWidth(4);
73          color(::active_ss == i ? DK_RED : MY_WHITE);
74          FrameRect(x1, y1, x2, y2);
75          LineWidth(1);
76    }
77
78
79    void
80    Zoomview :: calc_layout(int ss, int mod)
81    {
82            int i, j;
83            double modtot=0;
84            double cumx=0;
85
86            for (i=0, j=mod; i<modulesPerSS[ss]; i++, j++)
87                    modtot += modvar[j];
88
89            for (i=0, j=mod; i<modulesPerSS[ss]; i++, j++) {
90               modx[i] = cumx/modtot;
91               modw[i] = modvar[j]/modtot;
92
93               cumx += modvar[j];
94            }
95    }
96
97
98    void
99    Zoomview :: drawmodules(int ss, int mod)
100   {
101           int i, am, modstart;
102           double y1, y2, filly;
103
104           y1 = 0;
105           y2 = 1;
106
107           modstart = mod;
108
109           LineWidth(2);
110           for (i=0; i<modulesPerSS[ss]; i++, mod++) {
111                   // draw mqdule
112                   color(do_colors ? set_modcolor(mod)+mod%2 : DK_GRAY);
```

```
113                     FillRect(modx[i], y1, modw[i], y2);
114                     FrameRect(modx[i], y1, modw[i], y2);
115
116                     // draw subsystem fill average
117                     color(LT_GRAY);
118                     FillRect(modx[i], y1, modw[i], ssaverage[ss]);
119                     FrameRect(modx[i], y1, modw[i], ssaverage[ss]);
120
121                     // draw fill
122                     if (do_showfill) {
123                             color( do_colors ? MY_GRAY : LT_BLUE );
124                             filly=modvar[mod] ? modfill[mod]/modvar[mod]*y2 : 0;
125                             FillRect(modx[i], y1, modw[i], filly);
126                             FrameRect(modx[i], y1, modw[i], filly);
127                     }
128
129                     // outline
130                     if (do_modline) {
131                             color(MY_WHITE);
132                             FrameRect(modx[i], y1, modw[i], y2);
133                     }
134
135                     // set am to active module
136                     if (::active_module == mod)
137                             am = i;
138             }
139
140             // draw module names
141             if( do_modnames ) {
142                     double char_h = YStoU( getheight() ),
143                            ypos = .25;
144                     for(i=0, mod=modstart; i<modulesPerSS[ss]; i++, mod++)
145                       {
146                         color(active_module == mod ? DK_RED : MY_WHITE);
147                         DrawString( modx[i]+.5*modw[i], ypos, modnames[mod],
148                                     VxDrawCenter );
149                         ypos += 2*char_h;
150                         if(ypos>.75)
151                           ypos = .25;
152                       }
153             }
154
155             // draw active module
156             color(DK_RED);
157             FrameRect(modx[am], y1, modw[am], y2);
158             LineWidth(1);
159     }
 60
161     // this is a kluge
162     static GLXconfig glxConfig [] = {
163         { GLX_NORMAL, GLX_DOUBLE,   TRUE },
164         { GLX_NORMAL, GLX_COLORMAP, TRUE },
165         { GLX_NORMAL, GLX_ZSIZE,    GLX_NOCONFIG },
166         { 0, 0, 0 }
167     };
 68
```

```
169  Zoomview :: Zoomview(char const *name, Widget parent,
170                       int mybar_a) :
171
172          // default constructors
173          VxGLDrawingArea(name, parent, glxConfig),
174          VxMouseable( VxMotion ),
175          VxFunctionality(this)
176  {
177    mybar = mybar_a;
178    int ss = zoom[mybar].ss;
179
180          int i;
181          for (min_mod=i=0; i<ss; i++)
182                min_mod += modulesPerSS[i];
183          max_mod = min_mod + modulesPerSS[ss];
184
185
186    pub.Subscribe(this, ACTIVE_SS, ss);
187    pub.Subscribe(this, ACTIVE_MODULE);
188    pub.Subscribe(this, ACTIVE_SSVAR);
189    pub.Subscribe(this, ACTIVE_MODVAR);
190    pub.Subscribe(this, COLOR_CHANGE);
191    pub.Subscribe(this, FRAME_CHANGE);
192    pub.Subscribe(this, MODLINES_CHANGE);
193    pub.Subscribe(this, MODNAMES_CHANGE);
194    pub.Subscribe(this, SHOWFILL_CHANGE);
195    pub.Subscribe(this, ZOOMSS_CHANGE);
196  }
197
198  void
199  Zoomview :: DoExpose(int, int, int, int)
200  {
201    int mod=0;
202
203    color( BLACK );
204    clear();
205
206    int ss = zoom[mybar].ss;
207
208    UserCoord(-0.01, 1.01, -0.01, 1.01);
209
210    // draw subsystem
211    if (do_modules) {
212      calc_layout(ss, min_mod);
213      drawmodules(ss, min_mod);
214    }
215    else
216      drawss(ss, set_sscolor(ss));
217
218    // outline
219    LineWidth(4);
220    if (::active_ss == ss)
221      color(DK_RED);
222    else
223      color(MY_WHITE);
224
```

```
225     FrameRect(0., 0., 1., 1.);
226     LineWidth(1);
227
228     // ss name
229     //color(MY_WHITE);
230     //DrawString( .5, .5, ssnames[ss], VxDrawCenter );
231
232     swapbuffers();
233   }
234
235   void
236   Zoomview :: DoMouse(VxMouseActions /*what*/, int /*button*/,
237                       XEvent const * /*xevent*/)
238   {
239     double wmx, wmy;
240     int sschange, module, file;
241
242     // convert mouse position to world coordinates
243     wmx = XStoU( MouseX() );
244     wmy = YStoU( Height()-1 - MouseY() );
245
246     identify( wmx, wmy, &sschange, &module, &file);
247
248     if( sschange!=::active_ss )
249       {
250         ::active_ss = sschange;
251         pub.Changed(ACTIVE_SS, ::active_ss);
252       }
253     if( module!=::active_module )
254       {
255         ::active_module = module;
256         pub.Changed(ACTIVE_MODULE, ::active_module);
257       }
258
259   }
260
261
262   // user moved mouse in zoom window
263   void
264   Zoomview :: Update(int article, int part, void */*data*/)
265   {
266     switch(article) {
267
268     case ACTIVE_MODULE:
269           if( min_mod <=part && part<max_mod)
270                 ExposeMe();
271           break;
 72
273     case ACTIVE_SS:
274     case ACTIVE_SSVAR:
275     case ACTIVE_MODVAR:
276     case COLOR_CHANGE:
277     case FRAME_CHANGE:
278     case MODLINES_CHANGE:
279     case MODNAMES_CHANGE:
280     case SHOWFILL_CHANGE:
```

```
281     case ZOOMSS_CHANGE:
282       ExposeMe();
283       break;
284
285
286
```

bottom.h

```
1   #ifndef _BOTTOM_H
2   #define _BOTTOM_H
3
4   #include <Publication.h>
5   //#include <VxColor.h>
6   //#include <VxGC.h>
7   #include "VxDrawer.h"
8   #include <VxGLDrawingA.h>
9   #include <VxMouseable.h>
10  #include "seesys.h"
11
12
13  class Bottom :
14          public  VxGLDrawingArea,
15          public  VxDrawer,
16          private VxMouseable,
17          //private       VxColorManager,
18          //private       VxGC,
19          private Subscriber
20  {
21
22          void drawslider(int partitions);
23
24
25  // slider layout variables
26          double barx[MAX_MOD];     // x offset for subsystem bar in slider
27
28
29  public:
30
31
32          Bottom( char const *name, Widget parent );
33
34          void    DoExpose( int left, int top, int right, int bottom );
35          void DoMouse(VxMouseActions, int button, XEvent const *);
36          void Update(int article,  int /* part */,  void *);
37
38  };
39
40  #endif /* _BOTTOM_H */
```

left.h

```
1   #ifndef _LEFT_H
2   #define _LEFT_H
3
4   #include <Publication.h>
5   #include "VxDrawer.h"
6   #include <VxGLDrawingA.h>
7   #include <VxMouseable.h>
8
9
10  #define MAX_TOGGLES     10
11  #define MAX_SLIDERS     10
12
13  // *********** toggle vars *********
14  struct Toggle {
15          double x, y;
16          int *flag;
17          char *labels[2];
18          int  mypub;
19          Toggle(double x_a, double y_a, int *flag_s,
20                  char *offlabel, char *onlabel, int mypub_a);
21  };
22
23  struct Slider {
24    double x, y, h, w;
25
26    char *top_label;
27    char *left_toplabel;
28    char *left_bottomlabel;
29    char **barlabels;
30
31    // logical positions
32    int  min, max;
33    int *bar;
34
35    int  mypub;
36
37    Slider( double x_a, double y_a, double h_a, double w_a,
38            int min_a, int max_a, int *bar_a,
39            int mypub,
40            char *top_label_a,
41            char *left_toplabel_a=NULL,
42            char *bottom_toplabel_a=NULL,
43            char **barlabels_a=NULL);
44  };
45
46  class Left :
47          public  VxGLDrawingArea,
48          public  VxDrawer,
49          private VxMouseable,
50          private Subscriber
51  {
52
53          // this is for the animation!
54          XtIntervalId interval_id;
55          static void timeout(Widget w);
56
```

```
57          // my toggles
58          Toggle *toggles[MAX_TOGGLES];
59
60          // my sliders
61          Slider *sliders[MAX_SLIDERS];
62
63          void drawssstats();
64          void drawmodstats();
65          void set_ssstats();
66          double windsor(double low, double x, double high);
67          void identifyssstat(double wmx, double wmy, int *statbutton);
68          void identifymodstat(double wmx, double wmy, int *statbutton);
69          void drawframeslider(int frame);
70          void identifyframe(double wmx, double wmy, int *framepos);
71          void drawrowslider(int nrows);
72          void identifyrow(double wmx, double wmy, int *rowpos);
73
74          void drawslider(Slider *s);
75          int  identifyslider(double wmx, double wmy, Slider *s);
76
77          void drawtoggle(Toggle *t);
78          int  identifytoggle(double wmx, double wmy, Toggle *t);
79
80  public:
81
82
83          Left( char const *name, Widget parent );
84
85          void DoExpose( int left, int top, int right, int bottom );
86          void DoMouse(VxMouseActions, int button, XEvent const *);
87          void Update(int article,  int /* part */,  void *);
88
89  };
90
91  #endif /* _LEFT_H */
```

seeSYs.h

```
1   #ifndef _SEESYS_H
2   #define _SEESYS_H
3
4   #include <Publication.h>
5
6
7   #define MAX_ZOOM        10      /* max number of zoomed subsystems */
8   #define MAX_SS          50      /* max number of subsystems */
9   #define MAX_MOD         5000    /* max number of modules */
10  #define MAX_STRING      (8*1024)
11  #define PARTITIONS      4       /* number of slider partitions */
12
13  #define DASH_LINE       1       /* dash lines */
14
15  /* color definitions */
16  enum {
17          MY_BLACK= 0,
18          MY_WHITE= 1,
19          MY_CYAN = 4,
20          MY_GRAY,
21          DK_GRAY,
22          LT_GRAY,
23          DK_BLUE,
24          LT_BLUE,
25          DK_GREEN,
26          LT_GREEN,
27          DK_YELLOW,
28          LT_YELLOW,
29          DK_RED,
30          LT_RED
31  };
32
33
34
35  #ifndef MAX
36  #       define MAX(x,y)    ( (x)>(y) ? (x) : (y) )
37  #endif
38
39
40  // subsystem variables
41  extern char   **ssnames;          // subsystem names
42  extern int    nss;                // number of subsystems
43  extern double minssvar;           // largest subsystem variable
44  extern double maxssvar;           // largest subsystem variable
45  extern double maxssframevar;      // largest subsystem variable across all frames
46  extern int    modulesPerSS[MAX_SS];// number modules per subsystem
47  extern int    do_fill[MAX_SS];    // do fill stats exist
48
49
50  // module variables
51  extern char   **modnames;         // module names
52  extern int    nmod;               // number of modules
53  extern double maxmodvar;          // largest module variable
54  extern double maxmodframevar;     // largest module variable across all frames
55  extern int    domod_fill[MAX_MOD];      // do fill stats exist
56
```

```
57  // slider variables
58  extern int    cols[PARTITIONS]; // slider colors
59
60
61  // multiple variables
62  extern double *ssvar_all;       // all subsystem statistics
63  extern char   **ssvarnames;     // subsystem variable names
64  extern int    nssvars;          // number of subsystem variables
65  extern double *ssfill_all;      // all subsystem fill statistics
66  extern char   **ssfillnames;    // subsystem fill names
67  extern int    nssfills;         // number of fill variables
68  extern double *modvar_all;      // all module statistics
69  extern char   **modvarnames;    // module variable names
70  extern int    nmodvars;         // number of module variables
71  extern double *modfill_all;     // all module fill statistics
72  extern char   **modfillnames;   // subsystem fill names
73  extern int    nmodfills;        // number of fill variables
74  extern int    redraw_time;      // redraw time in milliseconds
75
76
77  // animation variables
78  extern char   **framenames;     // frame names
79  extern int    nframes;          // how many frames
80  extern int    frame_delay;      // milliseconds to delay btw frames
81
82  // zoom variables
83  struct ZOOM {
84    int ss;
85    int used;
86  };
87  extern ZOOM zoom[MAX_ZOOM];
88
89  // state variables
90  extern int do_animate;          // animate over frames
91  extern int do_animatelayout;    // zoom pending
92  extern int do_colors;           // colors for ss or modules
93  extern int do_fork;             // should we fork?
94  extern int do_modline;          // lines between the modules
95  extern int do_modules;          // show the modules
96  extern int do_modnames;         // show module names in zoomview
97  extern int do_showfill;         // show fill variable
98  extern int do_zoom;             // zoom pending
99
100
101 // active variables
102 extern double *ssvar;           // current subsystem variable
103 extern double *ssfill;          // current fill variable
104 extern double *modvar;          // current module variable
105 extern double *modfill;         // current fill variable
106 extern int    active_ssvar;     // which ss var is active
107 extern int    active_modvar;    // which module var is active
108 extern int    active_frame;     // which frame is active
109 extern int    active_ss;        // which subsystem is mouse on
110 extern int    active_module;    // which module is mouse on
111 extern int    active_nrows;     // how many rows in display
112
```

```
113  // subsystem layout variables
114  extern double ssxpos[MAX_SS];     / subsystems lower corners
115  extern double ssypos[MAX_SS];
116  extern double sswidth[MAX_SS];  // height and width
117  extern double ssheight[MAX_SS];
118  extern double ssmax[MAX_SS];    // max for each subsystem (ss animation)
119  extern double ssaverage[MAX_SS];// average fill accross ss
120
121  // module layout variables
122  extern double modxpos[MAX_MOD]; // module lower corners
123  extern double modypos[MAX_MOD];
124  extern double modwidth[MAX_MOD];        // height and width
125  extern double modheight[MAX_MOD];
126
127  // publications
128  extern Publication pub;
129
130  // events for pub
131  enum {
132          ACTIVE_SS = 1,          // active publications
133          ACTIVE_SSVAR,           // active subsystem variable
134          ACTIVE_MODVAR,          // active module variable
135          ACTIVE_MODULE,          // active module
136          ACTIVE_FILE,            // active file
137          ANIMATE_CHANGE,         // animate button pushed
138          COLOR_CHANGE,           // color toggle changed
139          FRAME_CHANGE,           // new frame var
140          MODLINES_CHANGE,        // modline toggle changed
141          MODNAMES_CHANGE,        // modline toggle changed
142          NROW_CHANGE,            // change in number of rows
143          REDRAWTIME_CHANGE,      // redraw time
144          SHOWFILL_CHANGE,        // do_showfill has changed
145          SPEED_CHANGE,           // user changed the speed
146          VAR_CHANGE,             // new active var
147          ZOOMSS_CHANGE,          // A zoomed window has moved
148          ZOOM_PENDING            // zoom pending
149  };
150
151  // X junk
152  #include <X11/Intrinsic.h>
153  extern XtAppContext app_context;
154  extern Widget toplevel;
155
156
157  // function prototypes
158  extern "C" void         // no name mangling
159  seesys(
160          int     *argc_ptr,
161          char    *argv[],
162          char    **ssnames_a,     // ss names
163          int     *nss_a,          // number of subsystems
164          char    **modnames_a,    // module names
165          int     *nmod_a,         // number of modules
166          \
167          // subsystem variables
168          double *ssvar_a,        // ss variables
```

```
169           int     *nssvars_a,       // number of ss variables
170           char    **ssvarnames_a,   // ss variable names
171           double *ssfill_a,         // all subsystem fill statistics
172           int     *nssfills_a,      // number of fill variables
173           char    **ssfillnames_a,  // subsystem fill names
174
175           // module variables
176           double *modvar_a,         // module variables
177           int     *nmodvars_a,      // number of module variables
178           char    **modvarnames_a,  // module variable names
179           double *modfill_a,        // all module fill statistics
180           int     *nmodfills_a,     // number of fill variables
181           char    **modfillnames_a, // module fill names
182
183           // animation frames
184           char    **framenames_a,   // frame names
185           int     *nframes_a,       // number of frames
186
187           int     *flags
188   );
189
190
191   double get_ssvar(int whichvar, int whichss);
192   double get_modvar(int whichvar, int whichmod);
193   void set_ssvar(int which, int nrows);
194   void set_modvar(int which, int nrows);
195   int  set_sscolor(int ss);
196   int  set_modcolor(int mod);
197   void layout(int nrows);
198   void modulelayout(int nrows);
199   void ssanimate_layout(int nrows);
200   void set_sslayout();
201   void set_maxssvar();
202   void set_maxmodvar();
203   void set_maxssframevar();
204   void set_maxmodframevar();
205   void zoomview_quitCB(Widget w,
206               caddr_t /*client_data*/, caddr_t /*call_data*/);
207
208
209   #endif /* _SEESYS_H */
```

sysview.h

```
1   #ifndef _SYSVIEW_H
2   #define _SYSVIEW_H
3
4   #include <Publication.h>
5   //#include <VxColor.h>
6   //#include <VxGC.h>
7   #include <VxGLDrawingA.h>
8   #include "VxDrawer.h"
9   #include "seesys.h"
10  #include <VxMouseable.h>
11
12
13  class Sysview :
14          public  VxGLDrawingArea,
15          public  VxDrawer,
16          private VxMouseable,
17          //private       VxColorManager,
18          //private       VxGC,
19          private Subscriber
20
21
22          void identify( double wmx, double wmy,
23                         int *ss, int *module, int *file);
24
25          void drawss( int which, int col);
26
27          void drawssname(int ss);
28
29          void drawmod( int which, int col);
30
31          void drawssboundaries(int ss);
32
33          void calc_fillaverage();
34
35          int active_ss;
36
37          int active_module;
38
39          int last_ss;
40
41  public:
42
43
44          Sysview( char const *name, Widget parent );
45
46          void    DoExpose( int left, int top, int right, int bottom );
47          void DoMouse(VxMouseActions, int button, XEvent const *);
48          void Update(int article,  int /* part */,  void *);
49
50  };
51
52  #endif /* _SYSVIEW_H */
```

zoomview.h

```
1   #ifndef  _ZOOMVIEW_H
2   #define  _ZOOMVIEW_H
3
4   #include <Publication.h>
5   #include "VxDrawer.h"
6   #include <VxGLDrawingA.h>
7   #include <VxMouseable.h>
8   #include "seesys.h"
9
10  class Zoomview :
11          public  VxGLDrawingArea,
12          public  VxDrawer,
13          private VxMouseable,
14          private Subscriber
15
16
17          void draw_stand_rel(
18              double x, double y, int which_rel, double size,
19              int toplabel,
20              Colorindex boundary, int boundary_thickness );
21          void drawss(int i, int col);
22          void drawmodules(int ss, int mod);
23          void calc_layout(int ss, int mod);
24          void identify(double wmx, double wmy, int *ss, int *module, int *file);
25
26          double modx[MAX_MOD];    // module x positions for zoom window
27          double modw[MAX_MOD];
28
29
30
31  public:
32
33          int mybar;
34          int min_mod;             // first module in my subsystem
35          int max_mod;             // one past last module in my subsystem
36
37
38          // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
39
40          Zoomview(char const *name, Widget parent, int mybar_a);
41
42          // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
43
44          void DoExpose(int left, int top, int right, int bottom);
45
46          void DoMouse(VxMouseActions what, int /*button*/,
47                       XEvent const * /*xevent*/);
48
49          void Update(int article,  int /* part */,
50                       void */* data*/);
51
52  };
53
54  #endif /* _ZOOMVIEW_H */
```

图 1

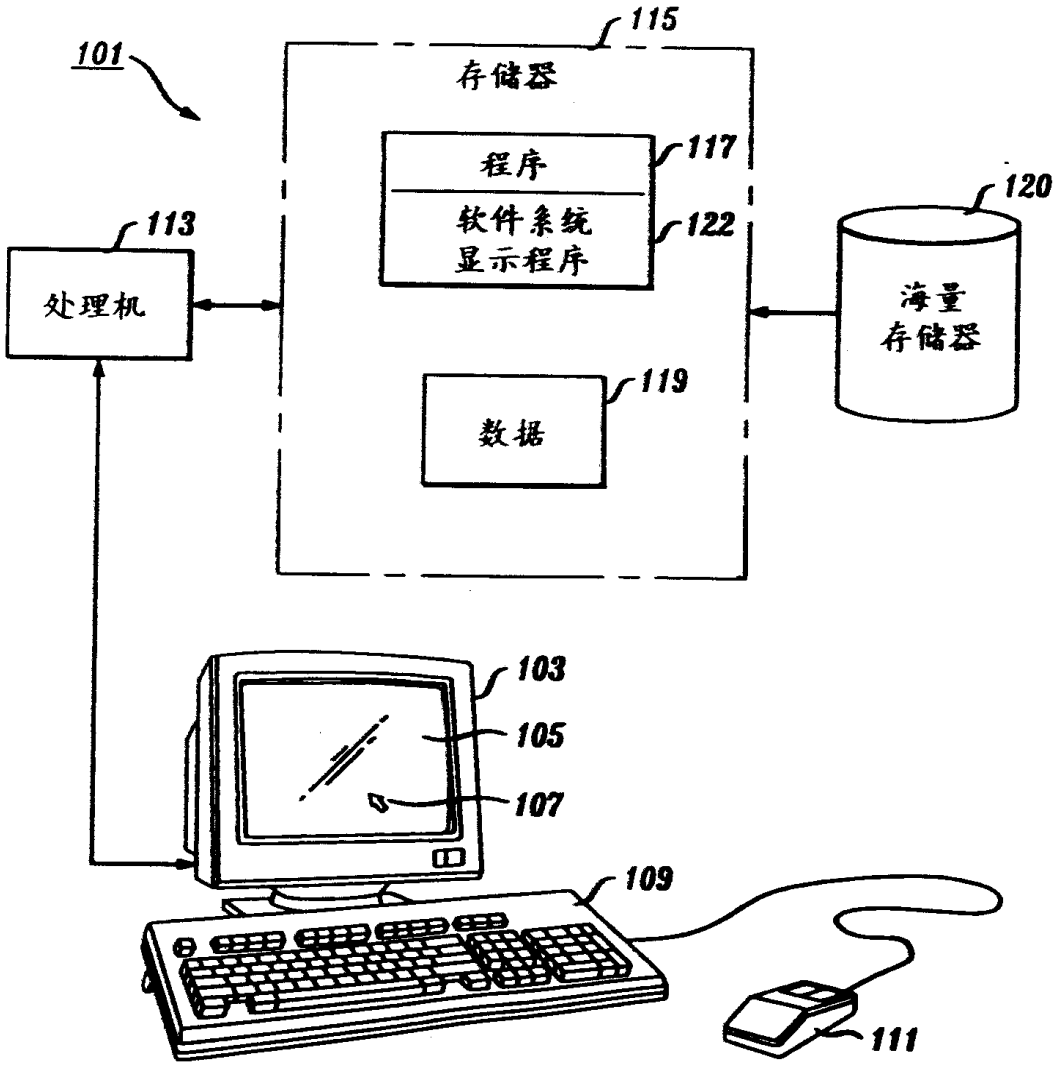

101
115
存储器
程序
117
软件系统
显示程序
122
120
113
处理机
海量
存储器
119
数据
103
105
107
109
111

图 2

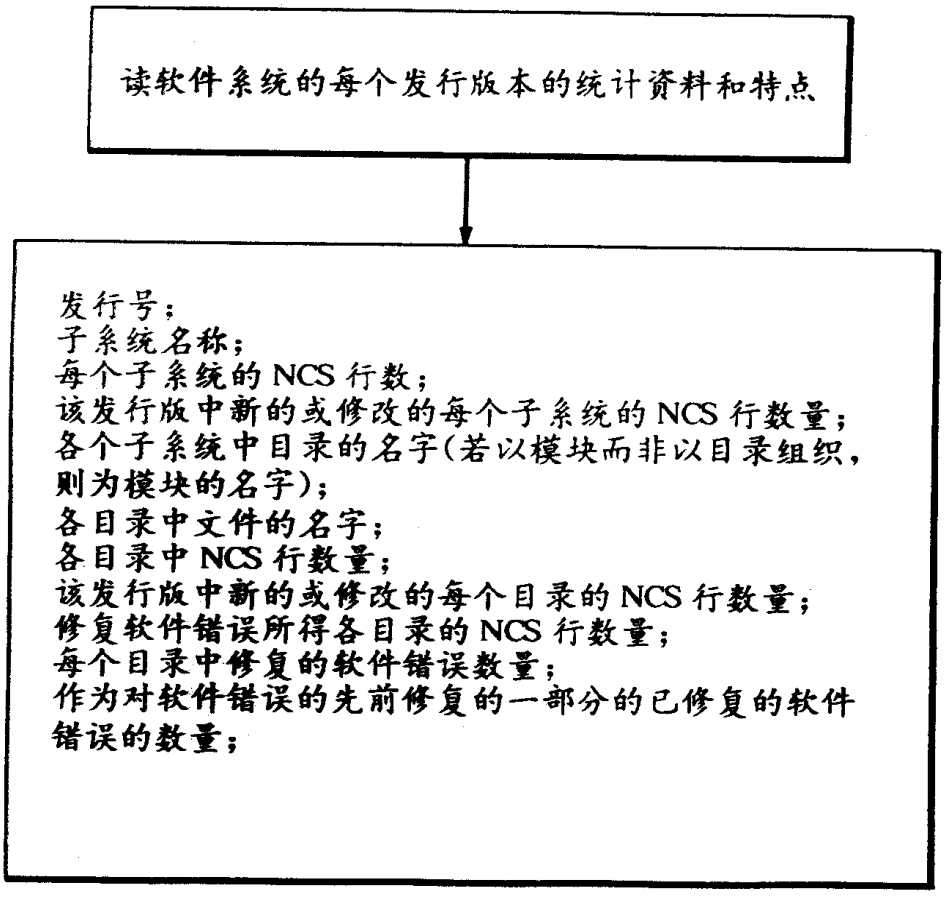

读软件系统的每个发行版本的统计资料和特点
发行号；
子系统名称；
每个子系统的NCS行数；
该发行版中新的或修改的每个子系统的NCS行数量；
各个子系统中目录的名字(若以模块而非以目录组织，则为模块的名字)；
各目录中文件的名字；
各目录中NCS行数量；
该发行版中新的或修改的每个目录的NCS行数量；
修复软件错误所得各目录的NCS行数量；
每个目录中修复的软件错误数量；
作为对软件错误的先前修复的一部分的已修复的软件错误的数量；

图 3

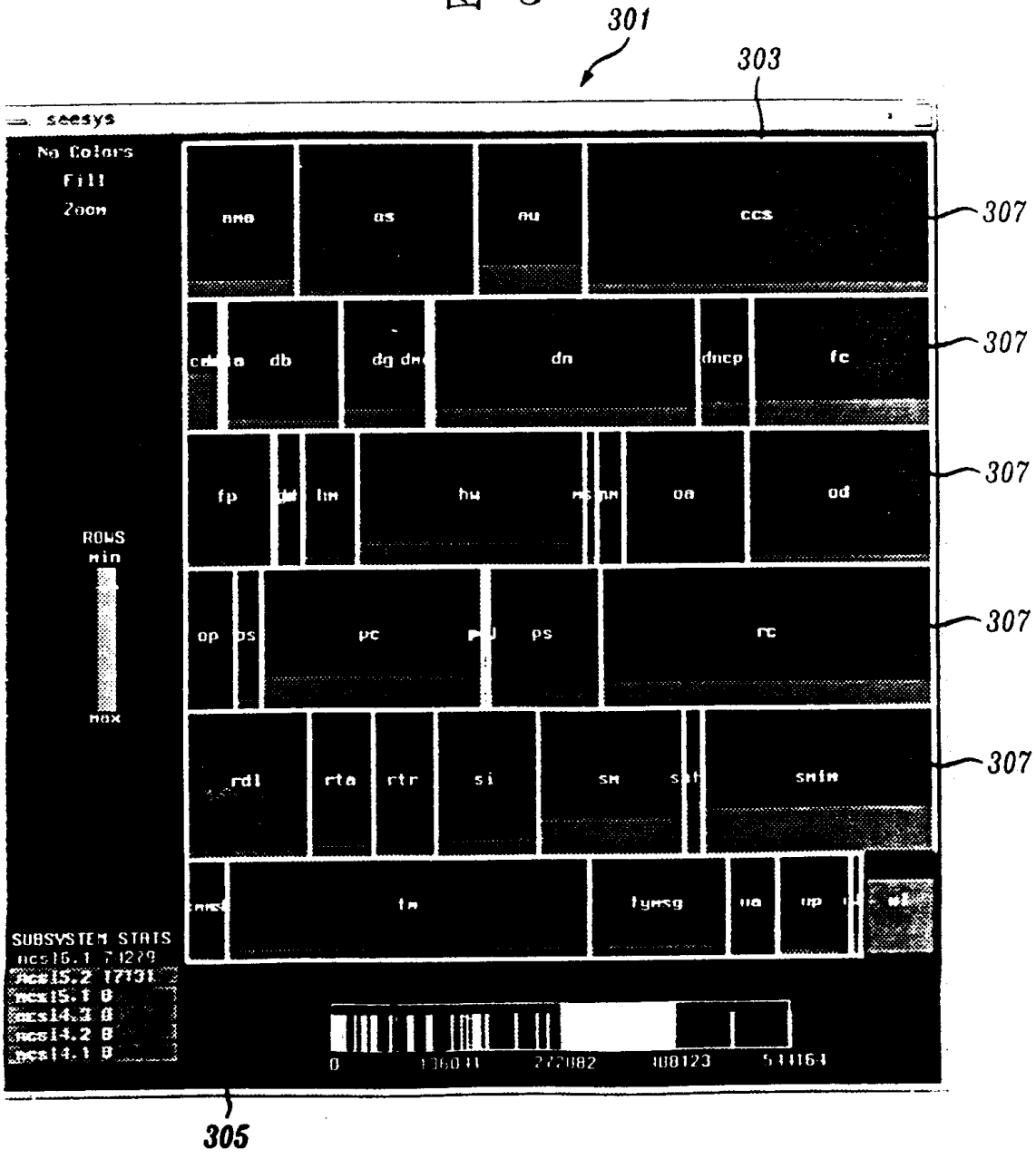

301
303
seesys
No Colors
Fill
Zoom
nma
as
au
ccs
307
db
dg
dn
dncp
fc
307
fp
lm
hu
oa
od
307
op
pc
ps
rc
307
ROWS
min
max
rdl
rta
rtr
si
sm
smin
307
fm
tymsg
up
SUBSYSTEM STATS
0
136011
272082
408123
544164
305

图 3A

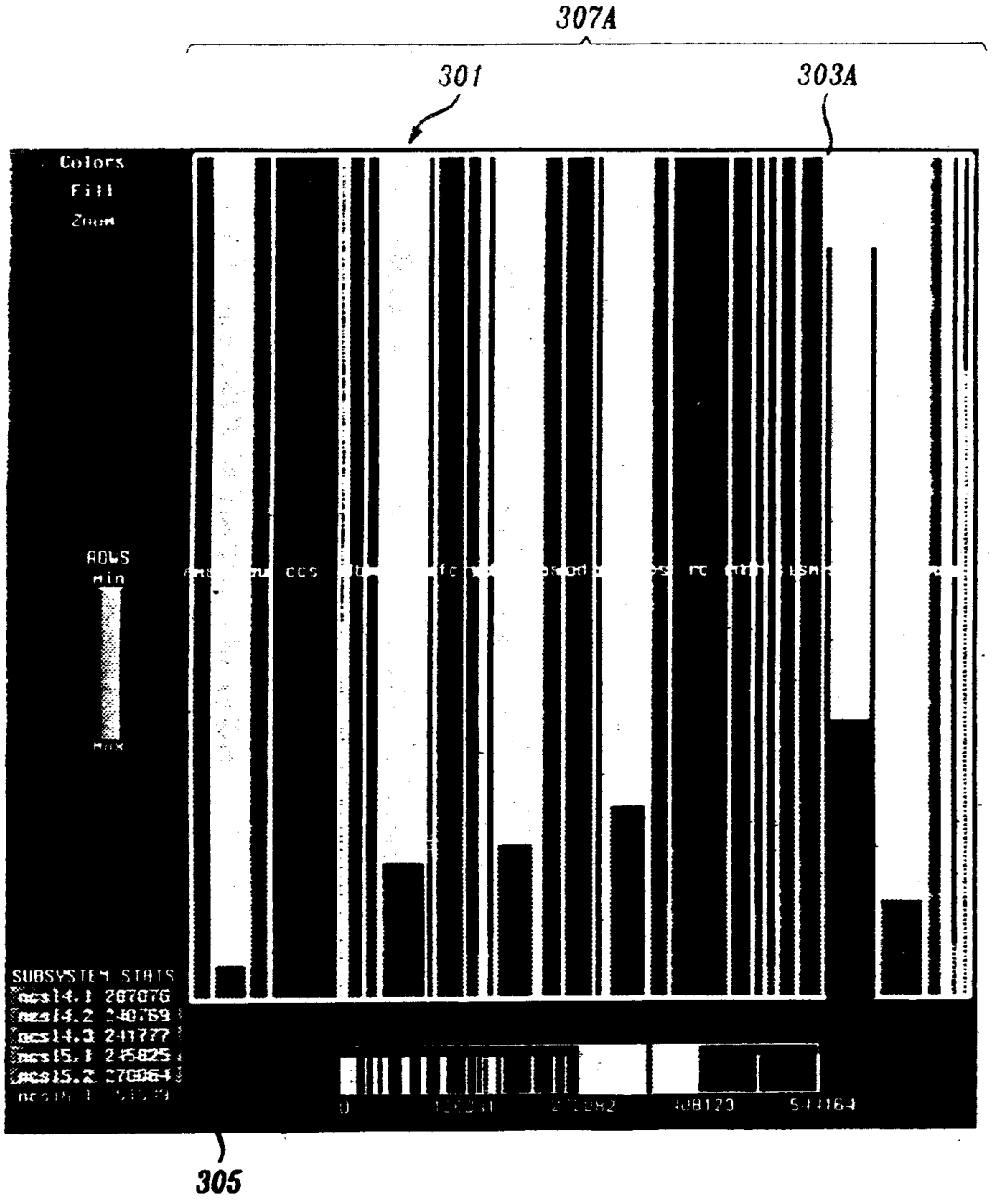


图 4

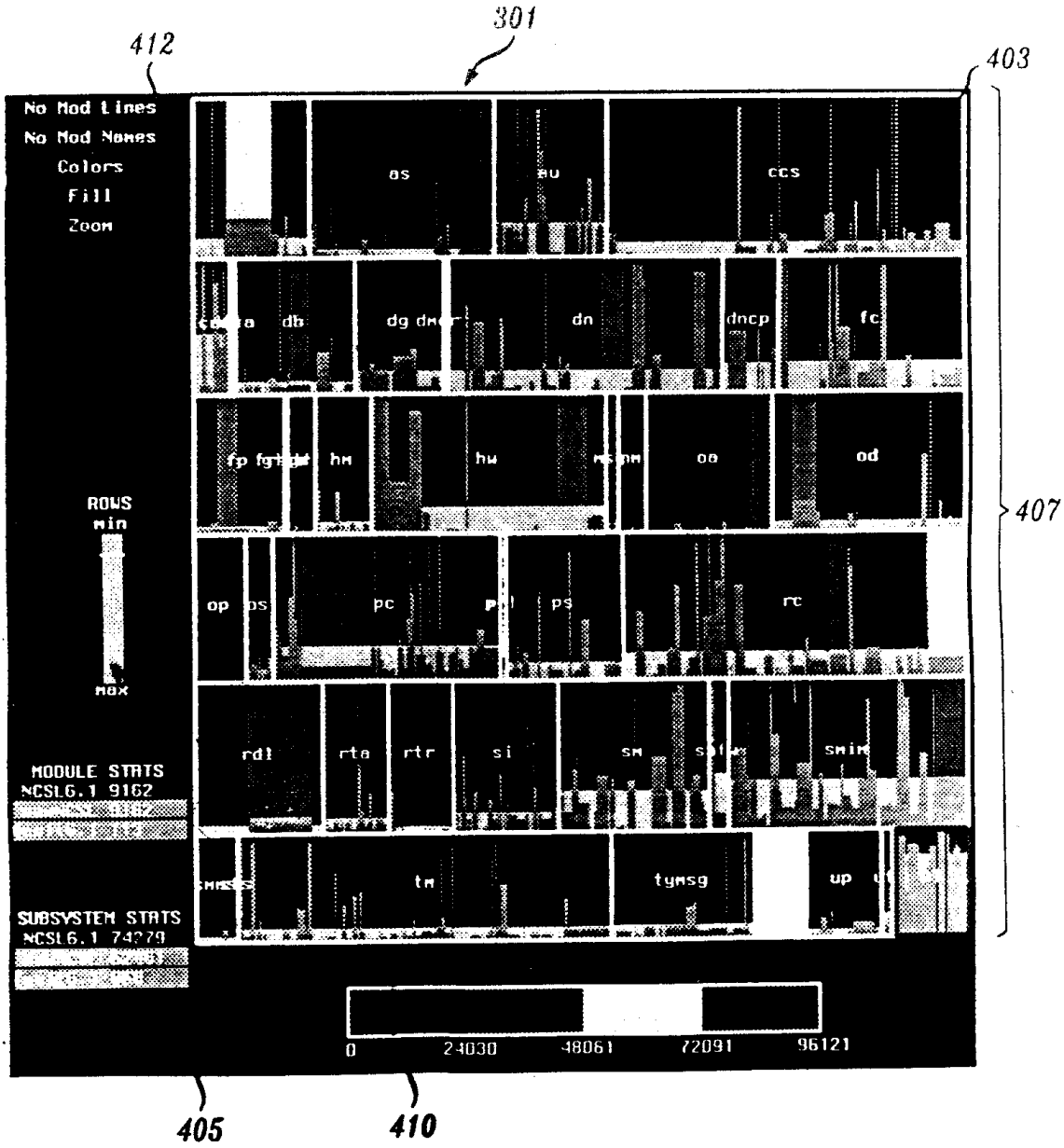

301
412
403
No Mod Lines
No Mod Names
Colors
Fill
Zoom
ROWS
min
max
MODULE STATS
NCSL6.1 9162
SUBSYSTEM STATS
NCSL6.1 74229
0
24030
48061
72091
96121
405
410
407

图 5

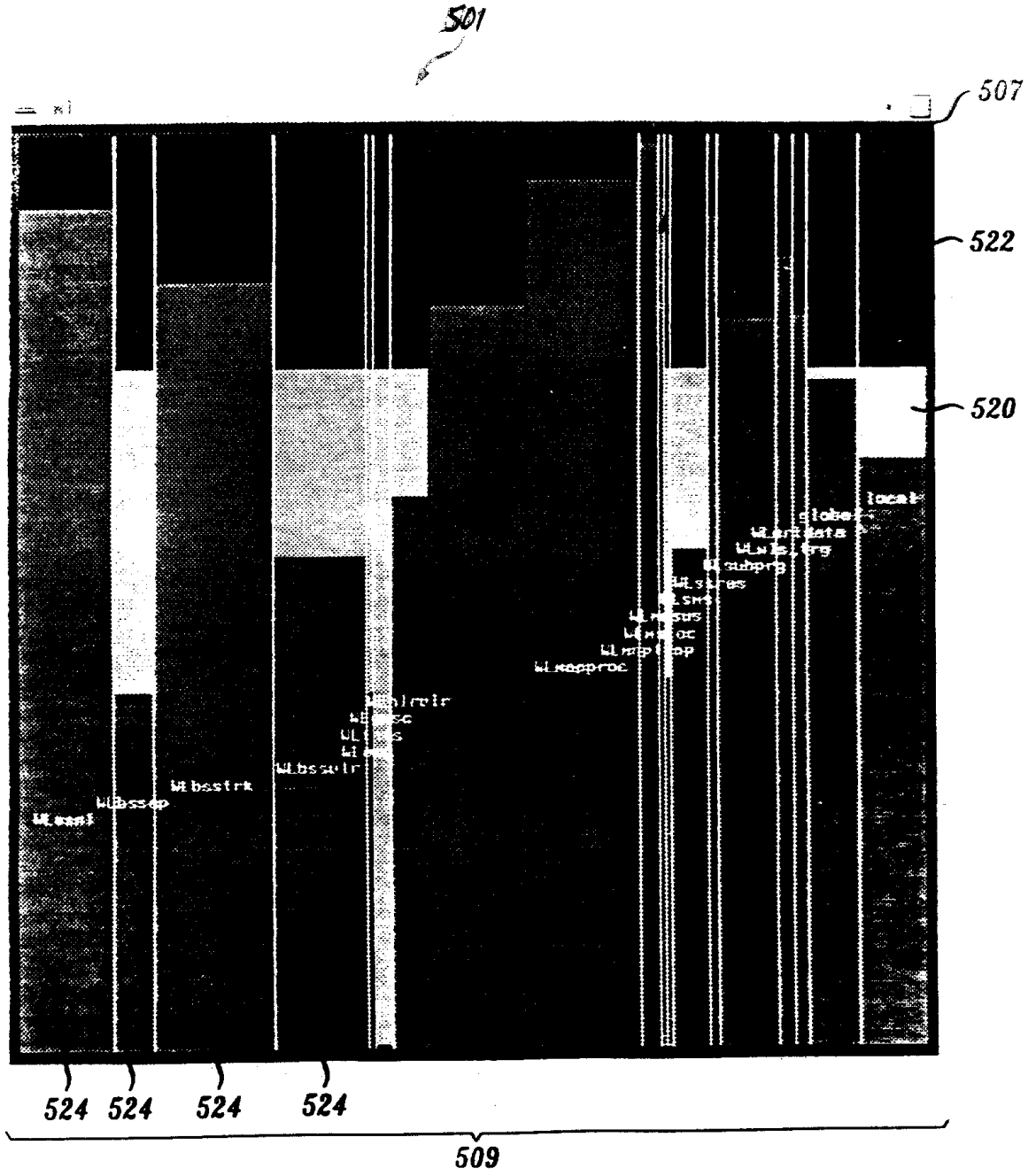


图 6

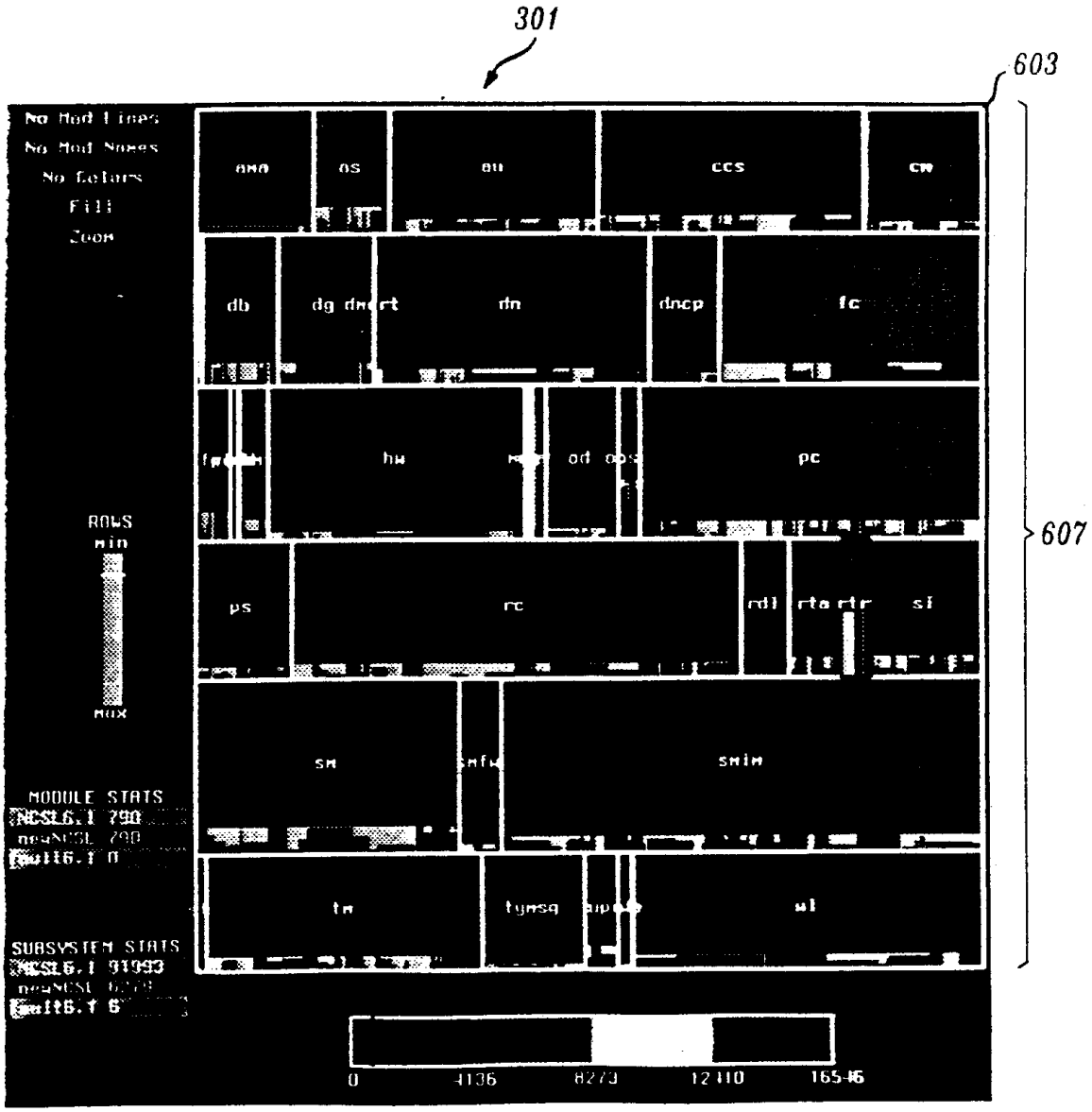

301
603
607
No Mod Lines
No Mod Names
No Colors
Fill
Zoom
amo
os
au
ccs
cw
db
dg
dn
dncp
fc
hw
od
pc
ps
rc
si
sm
smim
tm
tymsg
ul
ROWS
min
max
MODULE STATS
SUBSYSTEM STATS
0
4136
8273
12410
16546

图 7

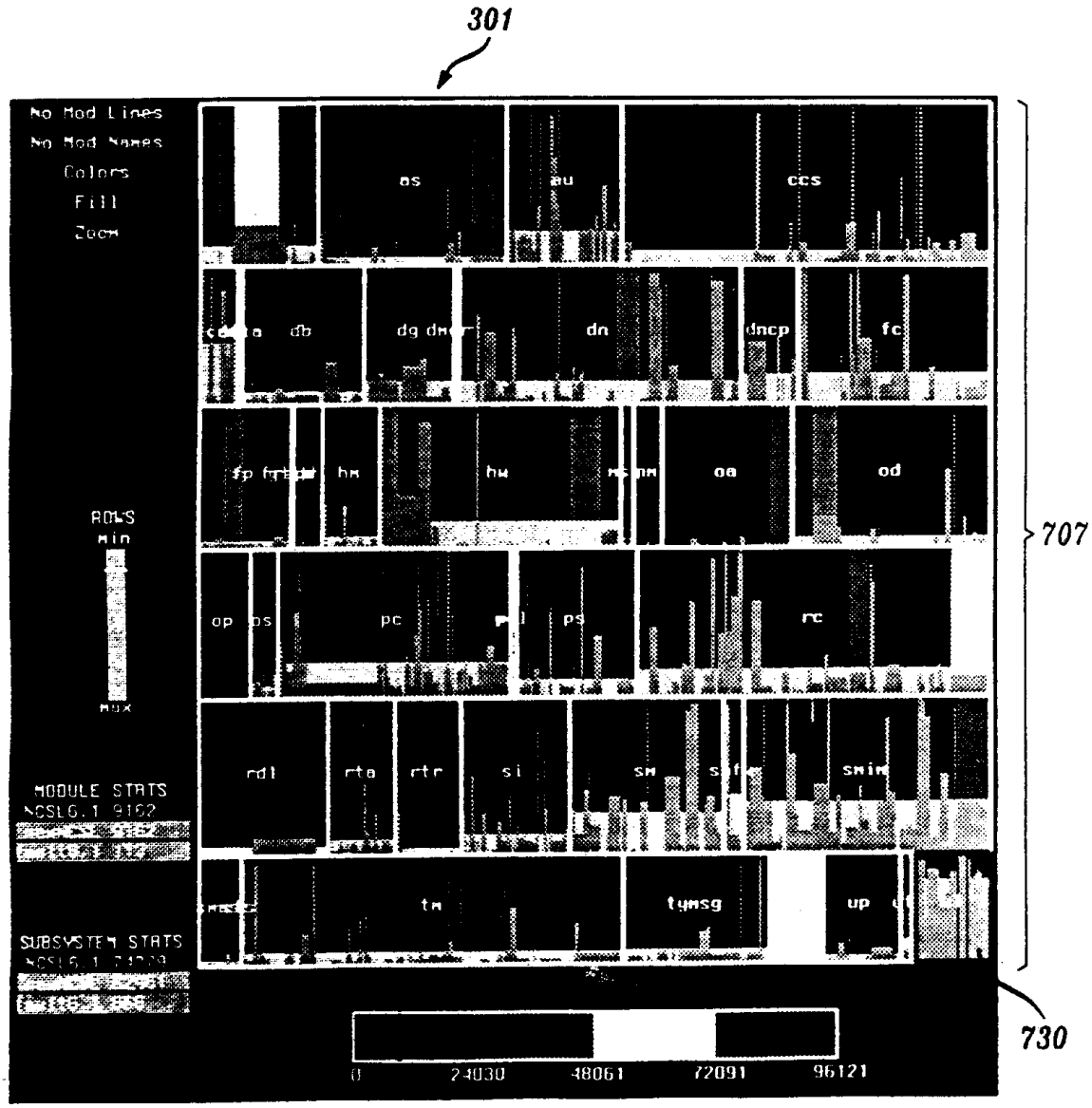

301
No Mod Lines
No Mod Names
Colors
Fill
Zoom
ROWS
min
max
MODULE STATS
SUBSYSTEM STATS
0
24030
48061
72091
96121
707
730

图 8

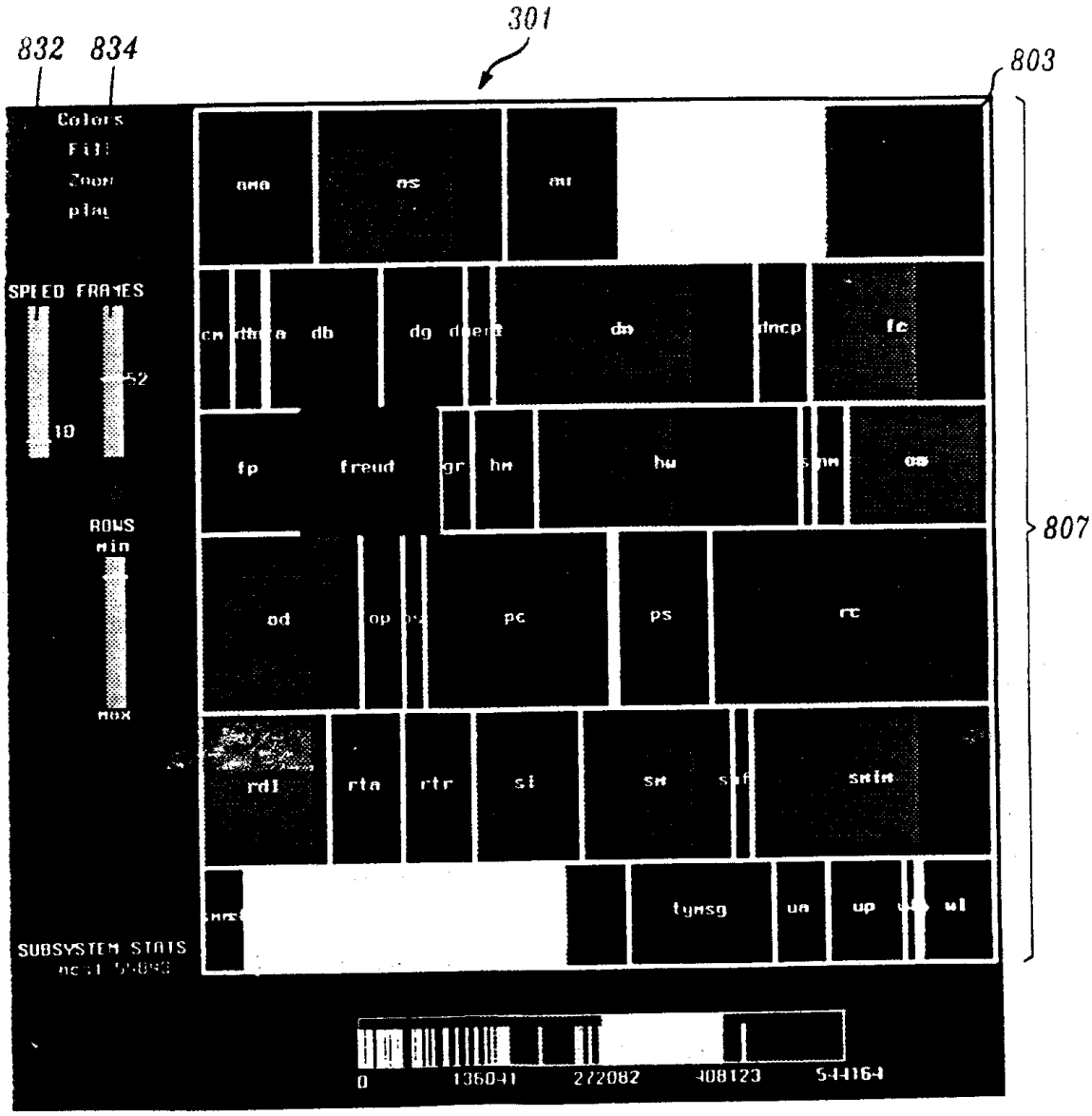

301
832
834
803
807
Colors
Fill
Zoom
play
SPEED FRAMES
52
10
ROWS
min
max
SUBSYSTEM STATS
0
136041
272082
408123
544164

図 9

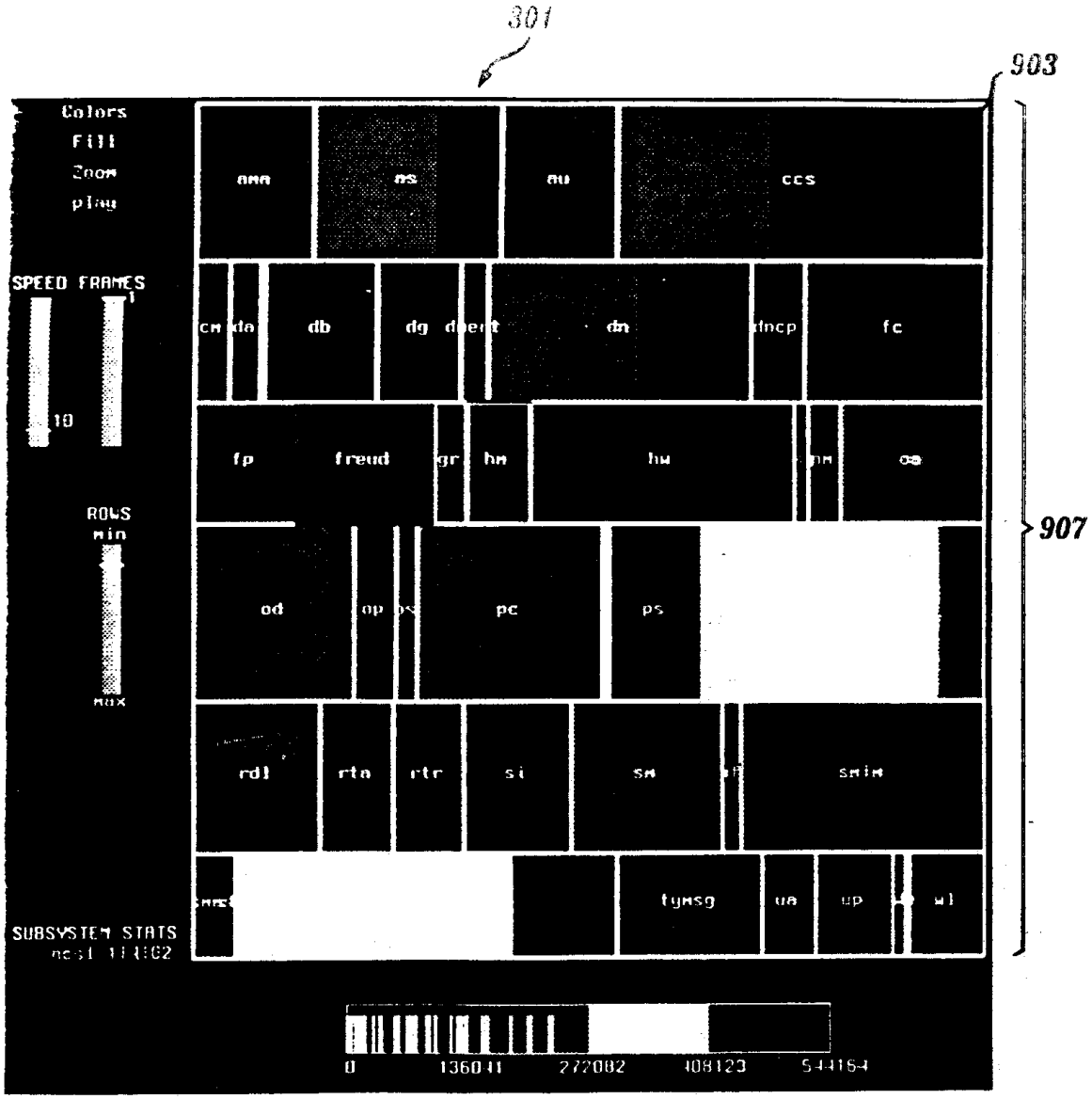
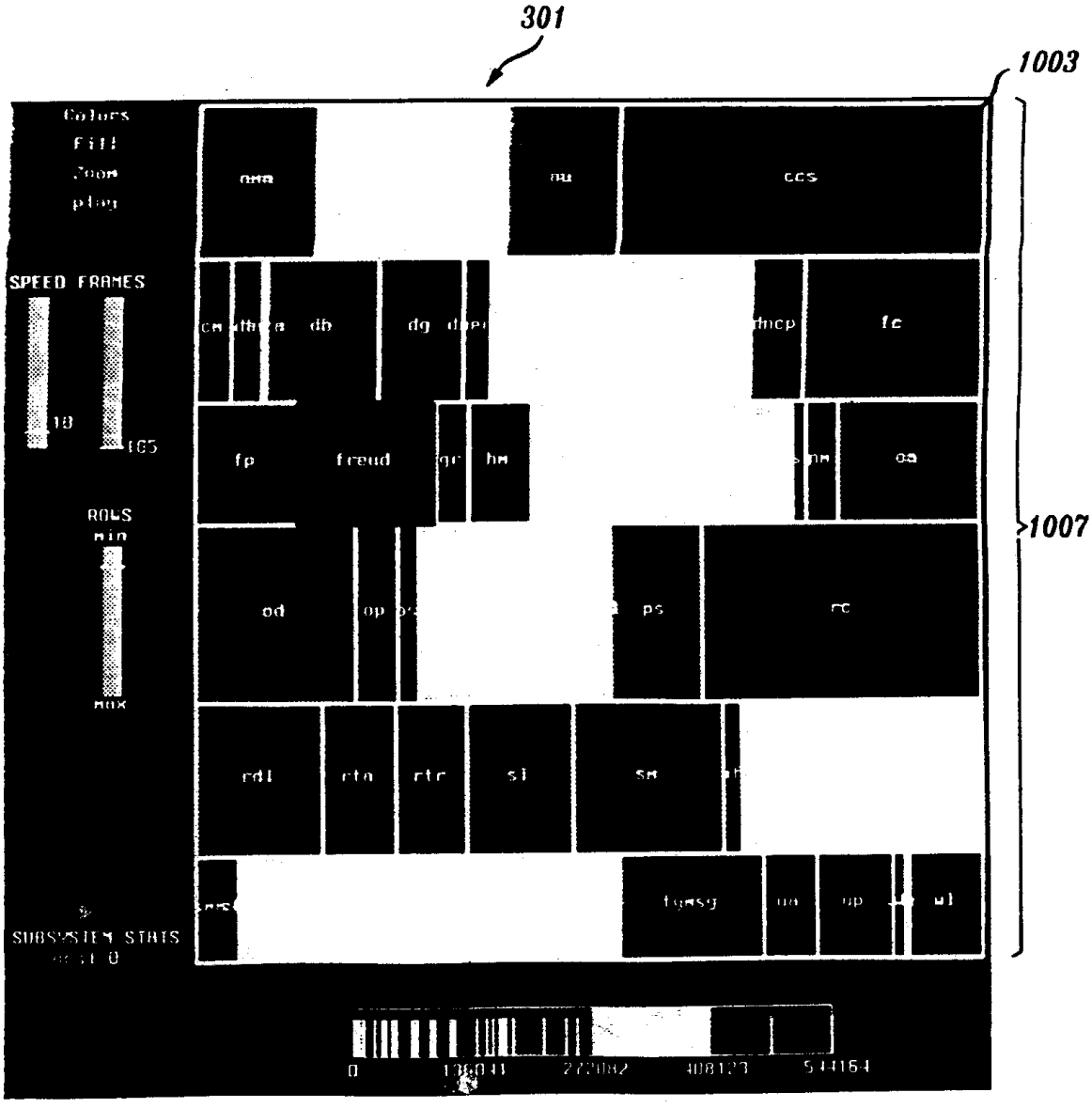

301
903
907
Colors
Fill
Zoom
play
SPEED FRAMES
10
ROWS
min
max
SUBSYSTEM STATS
ama
as
au
ccs
cm
da
db
dg
dn
dncp
fc
fp
freud
gr
hm
hw
od
op
pc
ps
rdl
rta
rtr
si
sm
smim
tymsg
ua
up
wl
0
136041
272082
408123
544164

图 10


301
1003
1007
Colors
Fill
Zoom
play
SPEED FRAMES
10
165
ROWS
min
max
SUBSYSTEM STATS
nmm
nu
ccs
cm
db
dg
dncp
fc
fp
freud
gr
hm
oa
od
op
ps
rc
rdl
rta
rtr
sl
sm
tgmsg
up
wl
0
272082
408123
544164