

remote updating of the logical flowsheets in each workstation.

In step **802**, a logical flowsheet in one of the workstations stores a new set of parameter values for a particular patient record. As illustrated in FIG. **9** and described above, the database server stores the new parameter values in the patient record. In addition, the database server in step **804** creates a database event that lists the changes to the patient flowsheet. In step **806**, the database server places the database event in the buffers of all flowsheet applications which have registered an interest in such database events. For example, with reference to FIG. **8**, assume that logical flowsheet **704A** stores a new parameter value for patient X. The database server **700**, in addition to storing the new parameter value, creates a database event that lists the new parameter value and places the database event in buffers **716** and **718** for logical flowsheets **706A** and **708A**, respectively, because logical flowsheets **706A** and **708A** pertain to patient X. The database event is not placed in buffer **714** because logical flowsheet **704A** originated the new parameter value and has already processed it. The database event is not placed in buffer **720** because corresponding logical flowsheet **710A** pertains to patient Y. It is assumed that logical flowsheets **706A** and **708A** have previously registered an interest in the new parameter value.

In step **808**, the database sends a database tickle to the logical flowsheets which have registered an interest in the new parameter value. The database tickle is a brief message that indicates to the logical flowsheet that its buffer contains a new database event for processing. The database tickle contains a sequence number, which indicates the chronological order of the database event and text, which may indicate the user that generated the database event.

The logical flowsheet in step **810** gets the database events, either immediately or at a later time when it is available to do so. The logical flowsheet may get the database events as part of a store operation, as shown in FIG. **9** and described above, or as part of a remote update operation, as shown in FIG. **11** and described below. After the logical flowsheet gets the database events, the database server deletes the database events from the buffer in step **812**.

The remote update operations for each flowsheet application are shown in FIG. **11**. Operations by the application task are shown on the left side of FIG. **11**, and operations by the logical flowsheet are shown on the right side of FIG. **11**. In step **850**, the logical flowsheet receives the database tickle from the database server, indicating the presence of a database event in the buffer for that logical flowsheet. In step **852**, the logical flowsheet calls a routine to receive database events. The sequence number of the database tickle is checked in step **854**. If the sequence number identifies a database event that has already been processed by the logical flowsheet during a store operation, no update is required and the operation is terminated.

Assuming that the database tickle contains a new sequence number, a callback to the graphic user interface is evoked in step **856**. The workstation is locked in step **858** so as to prevent further user interaction during the update process. In step **860**, a monolog box is displayed. The monolog box indicates that the flowsheet is being updated as a result of a change by another user or instrument. For example, the monolog box may state "Data for this patient has been updated by (list of users). The flowsheet will be refreshed to retrieve this data".

In step **862**, the logical flowsheet calls the routine to process database events and obtains all buffered database

events from its corresponding buffer in step **864**. If the buffer contains database events in addition to the one identified by the database tickle, a callback is evoked in step **866**, and the monolog box is updated in step **868** to reflect the additional users or instruments that have changed the patient flowsheet. Steps **866** and **868** are omitted when the buffer does not contain additional database events.

In steps **870** and **872**, the database events are processed as described above in connection with steps **744** to **748** in FIG. **9**. That is, the database events are processed in sequence, and a determination is made as to the necessity for updating the display screen. The display screen is updated if necessary in step **874** after processing all database events, and the workstation is unlocked in step **876**.

While the invention is described in terms of preferred embodiments in a specific system environment, those skilled in the art will recognize that the invention can be practiced, with modification, in other and different hardware and software environments within the spirit and scope of the appended claims.

What is claimed is:

1. A method for coordinating updates to a medical database in a medical information system, the method comprising the steps of:

entering a first data value for a record in said medical database at a first workstation and a second data value for said record at a second workstation, each of the workstations including a display screen;

permitting said first workstation to access said record in said medical database during data entry into said record at said second workstation and permitting said second workstation to access said record in said medical database during data entry into said record at said first workstation;

storing said first data value in said record in said medical database after completion of data entry for said record at said first workstation and storing said second data value in said record in said medical database after completion of data entry for said record at said second workstation, said first and second data values being stored in said medical database in dependent of the order in which they are entered at said first and second workstations; and

recording a correction history for said record, said correction history containing information as to the update of said record with said first data value and information as to the update of said record with said second data value.

2. A method as defined in claim 1 further including the step of locking said record to prevent access to said record by said workstations only during the steps of storing said first and second data values in said record in said medical database, thereby enabling concurrent data entry at said first and second workstations.

3. A method as defined in claim 1 further including the steps of defining database events including a database event for each of said first and second data values, placing database event data representative of said database events in buffers corresponding to said workstations, notifying said workstations of said database events, and transferring said database event data from said buffers to the corresponding workstations when each of said workstations requests such transfer.

4. A method as defined in claim 3 further including each of said workstations registering an interest in selected database event types, wherein the step of notifying said work-