Smart Medication Dispenser: Design, Architecture and Implementation

Pei-Hsuan Tsai, Tsung-Yen Chen, Chi-Ren Yu, Chi-Sheng Shih, *Member, IEEE*, and Jane W. S. Liu, *Fellow, IEEE*

***Abstract*—This paper presents the architecture and implementation of an automatic medication dispenser for users who take medications without close professional supervision. By relieving the user from the error-prone tasks of interpreting medication directions and administrating medications accordingly, the device can improve the rigor in compliance and prevent serious medication errors. By taking advantage of scheduling flexibility provided by medication directions, the device makes the user's medication schedule easy to adhere and tolerant to tardiness whenever possible. The medication scheduler and dispenser controller do this work collaboratively in an action-oriented manner. An advantage of this design is that new functions can be added and existing ones removed or revised with little or no need to modify the dispenser control structure.**

***Index Terms*—Action-oriented collaboration, embedded systems, smart device architecture.**

I. Introduction

ADVANCES in medical and pharmaceutical technologies over the years have led to more and more drugs that can cure or control previously fatal diseases and help people live actively for decades longer. The benefits of the drugs would be even more wondrous were it not for the high rate of preventable medication errors [1]–[4] that lead to thousands of serious adverse drug events, and billions of dollars in hospital cost each year in the U.S. alone. These alarming statistics have motivated numerous efforts in research, development and deployment of systems and tools for prevention of medication errors (e.g., [5]–[23]). Medication errors are known to occur throughout the medication use process of ordering, transcription, dispensing, and administration, with prescription errors introduced during ordering account for more than 50% of all errors. Computerized physician order entry (CPOE) systems [7]–[10] are now widely used in hospitals and clinics. Data available to date show that together with clinical decision support and electronic patient health and medication record (ePHR and eMAR) systems [5], [6], CPOE systems can help prevent up to 80% of prescription errors, i.e., 40% of all medication errors.

Next to prescription error, *administration errors* (i.e., errors due to failures to compliant to medication directions) are the most prevalent: They contribute 25–40% of all preventable errors. The smart medication dispenser described in this paper is designed to prevent this type of errors. It is primarily for the growing population of users who are elderly or have chronicle conditions but are well enough to live independently. Such a user may take many medications at home and work over a long period of time without close professional supervision.

Specifically, our smart dispenser is designed to eliminate two common causes of administration error: misunderstanding of medication directions and inconvenience of rigid medication schedules. Being almost fully automatic, the dispenser schedules individual doses of the user's medications under its care based a *medication schedule specification* (*MSS*). MSS is machine readable. It is compiled from the user's prescriptions and directions of over the counter (OTC) drugs by the user's pharmacist. The dispenser reminds the user at the times when some doses should be taken, monitors user's response to reminders, adjusts the medication schedule as needed when the user is tardy, and when noncompliance becomes unavoidable, sends notification in specified ways. Thus, the dispenser helps its user follow directions and stay compliant without having to understand the directions. This work is done collaboratively by the dispenser controller and medication scheduler in an action-oriented manner. Advantages of this design are generality and extensibility: As it will become evident in later sections that actions and action handlers can be added or removed to modify and configure the device with little or no need to change the control structure of the dispenser.

The remainder of the paper is organized as follows. Section II presents an overview of tools that provide support for smart dispensers and compare our dispenser with other medication usage assistance devices. Section III presents key assumptions that must be valid for our dispenser to work, illustrates its operations by a user scenario and describes the timing and dosage constraint parameters defined by MSS. Section IV presents the architecture and major components of the dispenser. The prototype software of the dispenser has an action-oriented structure. Section V describes the interface and communication flow for action-oriented collaboration. Section VI describes the behavior specification and software structure of the dispenser and presents an illustrative example. Section VII describes how the dispenser has been evaluated for the purpose of ensuring that the device works as intended. Section VIII summarizes the paper and discusses future work.

Manuscript received February 26, 2010; revised June 29, 2010; accepted June 29, 2010. Date of publication September 27, 2010; date of current version February 18, 2011. This work was supported in part by Taiwan Academia Sinica thematic project SISARL.

P.-H. Tsai, T. Y. Chen and C. Y. Yu are with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan (e-mail: peipei@iis.sinica.edu.tw; yen@iis.sinica.edu.tw; rayswin@gmail.com.

C. S. Shih is with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan (e-mail: cshih@csie.ntu.edu.tw).

J. W. S. Liu is with the Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan (e-mail: janeliu@iis.sinica.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2010.2070970

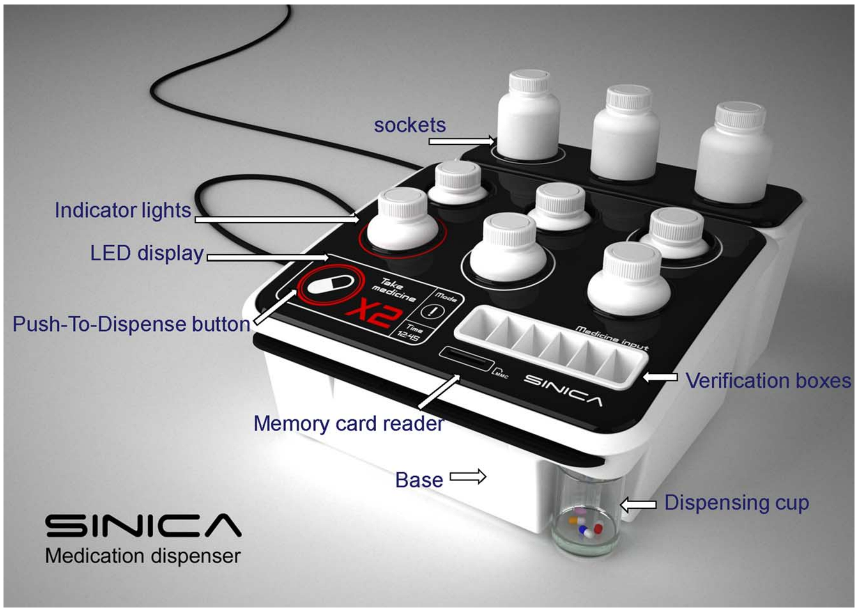


Fig. 1. Smart dispenser.

II. Related Works

There are a large variety of medication administration assistance devices for nonprofessional users. Most stand-alone devices (e.g., [16]–[18]) available today are manual: the user must load the individual doses of medications into the device, understand their directions and programs the device to send reminders accordingly. This manual process frequently introduces errors, which our dispenser is designed to prevent.

Like schedules used by our dispenser, medication schedules used by automatic devices and scheduling tools such as MEDICATE Tele-assistance System [20], [21] and Magic Medicine Cabinet [22] can be adjusted to compensate for user tardiness and condition changes. The adjustments are made by care providers who monitor and supervise the user via Internet, however. Those devices are suited for users who need close professional supervision and fully integrated health care services. In contrast, our dispenser is a stand-alone tool, capable of making schedule adjustments permitted by existing prescriptions. It is for individuals who are well and hence do not want to incur the cost of continuous monitoring and care and consequent loss of privacy and independence.

A typical user is likely to be under the care of multiple physicians and given prescriptions ordered via the physicians' CPOE systems [7]–[10]. The CPOE systems may be independent; they may fail to account for interactions between medications ordered by different prescriptions and the OTC medications taken by the user. The prescription authoring tool described in [15] is designed to help user's pharmacist detect and eliminate this kind of error. Another important function of the tool is the generation of medication schedule specifications (*MSS*) that guide the operations of the dispensers. The tool first merges all of user's prescriptions and OTC medication directions. Based on information from drug libraries (e.g., [14]), the tool checks for drug interactions that require further attention and assists the pharmacist to resolve the problem. It then translates the merged directions thus generated into a MSS written in XML language for the user's dispenser. The tool also makes sure that all the constraints defined by MSS are *feasible*, i.e., there is at least a schedule meeting all the constraints. Several algorithms for this purpose are described in [23]–[26].

III. Backgroundand Assumptions

For any automatic medication dispenser serving a user to be effective in prevention of medication errors, the following assumptions must be valid. These are our assumptions.

1) The tool manages all the medications of the user.
2) The medication schedule specification (MSS) used to guide the operations of the dispenser is generated based on a complete and current medication record of the user.

Although the dispenser does not handle food, it must schedule meals and snacks along with medications when food interferes with some of the user's medications.

A. A Use Scenario

When the user goes to fill a new prescription or purchase some OTC drugs, the pharmacist uses a prescription authoring tool [15] to process all prescriptions and medication directions of the user and generate a MSS for the user's dispenser. The pharmacist gives to the user the MSS and new supplies of medications in containers. Each container holds a medication identified by the RFID in the tag attached to the container. The MSS is stored in a memory card or a flash disk.

Fig. 1 shows the smart dispenser. The dispenser has on its base a number of sockets, an indicator light around each socket, one or more reminder devices (e.g., an audio alarm, a flashing light, a phone, etc.), a LED display, a Push-To-Dispense (PTD) button, verification boxes, a dispensing cup and a memory card reader. The RFID reader for reading tags on containers sits inside the base. Containers holding medications taken by the user are plugged in sockets. There is a switch inside the base for each socket. The switch is closed when a container is plugged in the socket; otherwise it is open.

1) Set Up: To put new supplies under the care of the dispenser, the user plugs the MSS disk into memory card reader of the dispenser and the new containers into the empty sockets in any order. The dispenser picks up from the MSS disk the updated medication list and constraints for scheduling the new medications along with existing ones. Whenever the dispenser controller senses that the state of the switch for a socket (say socket k) changes from open to close, it commands the RFID reader to read the tags on all containers in sockets. Upon discovering a new id (say M), it creates and starts to maintain the id-location mapping (M, k) for the new medication and locks the container in socket. The controller disallows multiple containers being plugged in at the same time. When the controller senses that the user has simultaneously plugged in more than one container, it prompts the user to remove the containers involved and plug them back again one at a time.

2) Normal Operations: Set up completes and normal operations commence when the dispenser base holds a container and has the id-location mapping for every medication listed in the MSS. The dispenser first computes a medication schedule. The schedule specifies the time instants (hereafter, referred to as *dose times*) and dose sizes of medications to be taken. Shortly before each dose time, the dispenser uses the reminder devices to tell the user to come to retrieve and take medication(s). In response to the reminder, the user reports to the dispenser by pushing the PTD button.

Because the amount of time the user takes to respond to a reminder may vary widely, the dispenser updates the dose size of each medication due to be taken when the PTD button is pushed. For each medication due to be taken, the dispenser lights up the indicator light around the socket holding the container for the medication and unlocks the socket to allow the removal of the container. When the user picks up the container, the LED display shows the dose size to be taken at the time. After the user retrieves the dose from the container and puts the container back to the socket, the dispenser locks the container in place again. The dispenser and the user repeat this process if more medication(s) is scheduled to be taken at the time.

A dispenser with the verification capability is equipped with a camera to capture the image of objects placed in verification boxes. The user needs to put each retrieved dose in a verification box. Once there, the dispenser checks visually whether the retrieved dose size is correct. It uses the text display to instruct the user when correction is necessary, and when there is no error, locks the returned container in place and drops the medication into the dispensing cup.

B. Medication Schedule Specification

Again, the dispenser computes and adjusts the user's medication schedule based on the constraints given by the user's MSS. The MSS contains a section for each medication. Table I summarizes the key elements in the section. The first part gives the information required by the dispenser to administrate the medication, including the name or id (say it is M) of the medication and the *duration* for the user is to be on the medication. The dispenser uses the same time resolution for all medications. The medication comes in granules of size g; dose size parameters of the medication are given in terms of integer multiples of g. This part also contains a human readable description of the purpose of the medication. The LED display of the dispenser is too small for effective display of but a few words (e.g., "Rosuvastatin is for control of cholesterol") from the description, however. The user can use any device with a memory card reader (or USB port) to read the full description, or have the pharmacists print out the description for him/her.

TABLE I
SECTION OF MSS FOR MEDICATION M

- *M*: Name of the medication
- *g*: Granularity
- $[T_{min}, T_{max}]$: Minimum and maximum durations
- Description of purpose and other relevant attributes
- Dosage Parameters (DP)
    1. $[d_{min}, d_{max}]$: Nominal minimum and maximum dose sizes
    2. $[s_{min}, s_{max}]$: Nominal minimum and maximum separations
    3. (B, R): Maximum intake over a specified time interval given by budget *B* and replenishment delay *R*
    4. (L, P): Minimum intake over a specified time interval given by lower bound *L* and interval length *P*
    5. $[D_{min}, D_{max}]$: Absolute minimum and maximum dose sizes
    6. $[S_{min}, S_{max}]$: Absolute minimum and maximum separations
    7. Non-compliance event types and corresponding actions.
- Special Instructions (SI)
    1. *N*: Name of an interferer
        a. Change list
        b. $\sigma_{min}(M, N)$: Minimum separation from *M* to *N*
        c. $\sigma_{min}(N, M)$: Minimum separation from *N* to *M*
    2. *Ni*: Name of another interferer

        . . .

The dosage parameters (*DP*) part specifies constraints on dose size and separation (i.e., the length of time interval between any two consecutive doses) for scheduling the medication when the medication is taken alone. Specifically, lines 1 and 2 give *nominal dose size* range $[d_{\min}, d_{\max}]$ and *nominal separation range* $[s_{\min}, s_{\max}]$. Lines 3 and 4 specify the *supply rate* (B, R) and *demand rate* (L, P): The former says that the *intake* (i.e., the total size of all doses) within any time interval of length R must be no more than B. The latter says that the intake must be no less than L in any interval of length P. Normal schedules are computed based on these parameters.

The DP part may also include *absolute dose size range* $[D_{\min}, D_{\max}]$ and *absolute separation range* $[S_{\min}, S_{\max}]$. These constraints are hard. By making absolute ranges wider than the corresponding nominal ranges, the direction allows some flexibility in scheduling. Deviations from normal schedule may occur, mostly due to user's tardiness, and some may lead to violations of hard constraints. The dispenser treats each violation of a hard constraint as a *noncompliance event* and is required to take some specified action(s) (e.g., measure blood pressure and contact a care taker). Specifications on the actions required to handle each type of noncompliance events are included in the MSS. This aspect is out of the scope here.

The third part of the section for medication M contains special instructions (*SI*) for scheduling M when some of user's medications interact with M or when M interacts with food. We refer to a medication (or food) that interacts with M to the extent as to require some changes in how M is to be administered as an *interferer* of M. The SI part of M has an entry for

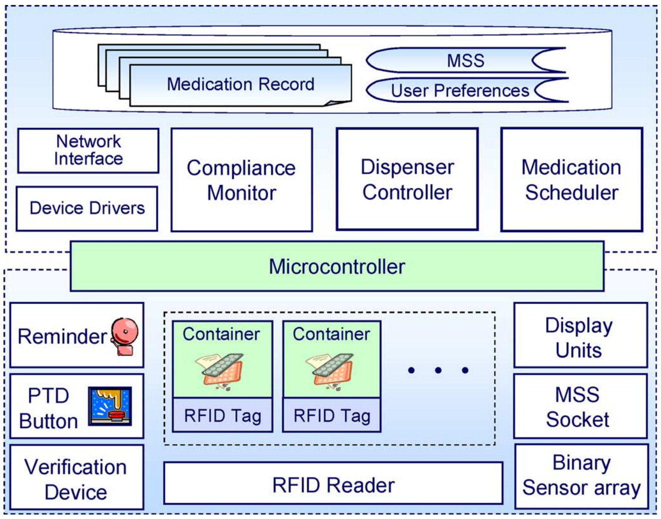


Fig. 2. Dispenser architecture.

each of its interferers. The *change list* in the entry for an interferer (say N) specifies changes in dosage parameters of M that are in effect as long as the user is on both M and N. The entry of an interferer N may also define additional separation constraints, each of which specifies a required time separations between each dose of M and any dose of the interferer N. Table I lists only the *minimum separation* $\sigma_{\min}(M, N)$ *from the medication to interferer* for each dose M scheduled before any dose of N and the *minimum separation* $\sigma_{\min}(N, M)$ *from the interferer to the medication* for each dose of N scheduled before some dose of M.

Table I leaves off precedence constraints and additional separation constraints due to medication interaction, as well as parameters indicating user preferences. These constraints are discussed and illustrated in [26].

IV. Dispenser Architecture

Fig. 2 shows the architecture of the smart dispenser. The dotted box at the bottom encircles its hardware components. We will return shortly to provide specifies and details on them.

A. Major Software Components

The dotted box on the top encircles software components. The current version of the prototype is implemented in C programming language and is released under GPL license. It is multi-threaded and event driven and runs as an application on a desktop PC running Microsoft Windows XP. It can be easily ported to an embedded platform like Windows CE and, in general, to any operating system that supports threads and allows threads to wait for events and timer expiration.

The dispenser controller (or controller for short) and the medication scheduler (or scheduler) are central to dispenser operations. The controller extracts from the MSS file the information needed for scheduling, dispensing and compliance monitoring and puts the information in a structure convenient for internal use. The scheduler computes an initial medication schedule immediately after set up. It is also responsible for adjusting the schedule when the user is tardy to prevent noncompliance if at all possible and for determining the actions to carry out when a noncompliance event occurs. The heuristic scheduling algorithms used by the scheduler and the underlying resource model are described in [22]–[26].

The medication scheduler has full knowledge of what medication administration related actions should be done at what instants of time, but has no knowledge of time. In contrast, the controller is responsible for keeping track of time, informing the scheduler the arrivals of time instants for actions specified by the scheduler, and overseeing the execution of the actions. The controller also monitors conditions of all components and handles events indicating the occurrences of conditions (e.g., insufficient medication supply and broker indicator light) that warrant user attention and recovery actions.

The top dotted box also shows compliance monitor, network interface, user preferences, and medication (administration) record. Due to space limitation, we will not elaborate further about them. For sake of discussion here, it suffices to note that the compliance monitor is responsible for generating and sending notifications in specified manners when invoked by the controller to do so. The version of the dispenser implemented to date relies on a local alarm and a dial-up connection for this purpose. An enhanced version of the dispenser can be configured to use Internet and to capture user preference and record user behavior in order to better serve the user.

B. Hardware Components and Driver Interface

The data rate between the host and each hardware device is very low. For this reason, we make all hardware devices, except the MSS disk, share a RS232 connection and use a microcontroller to support and manage their communication to and from the host via RS232. Fig. 2 shows this arrangement. The microcontroller forwards commands issued by the device driver of each hardware device to the device, and the device driver abstracts low-level instructions of the device into general driver functions.

In general, drivers of hardware components provide the dispenser controller with two kinds of facilities: hardware control and event notification. The former consists of commands which the controller can call to request services. The latter is the primary means of communication from hardware to controller. As an example, Fig. 3(a) shows the logic diagram of a binary sensor array (BSA), which implements the array of switches illustrated by Fig. 3(b). The BSA driver provides the controller with a command to reset all switches and clear the array and a bitmap to indicate the current states of all the switches.

Specifically, the BSA driver communicates with the dispenser controller via three events. The driver sets event OBJECT_IN (or OBJECT_OUT) when the switch in a socket changes state from 0 to 1 (or 1 to 0) indicating that a container is just plugged in (or removed from) the socket. In response the controller calls the handler GetPluggedInSocket() (or GetPluggedOutSocket()) to get the index (i.e., location) and current state of the socket. The driver sets STATE_CHANGE event when more than one switch change state. The controller can determine the switches involved and their current states by the function GetSensorStates (char* Buffer) to get the bitmap.

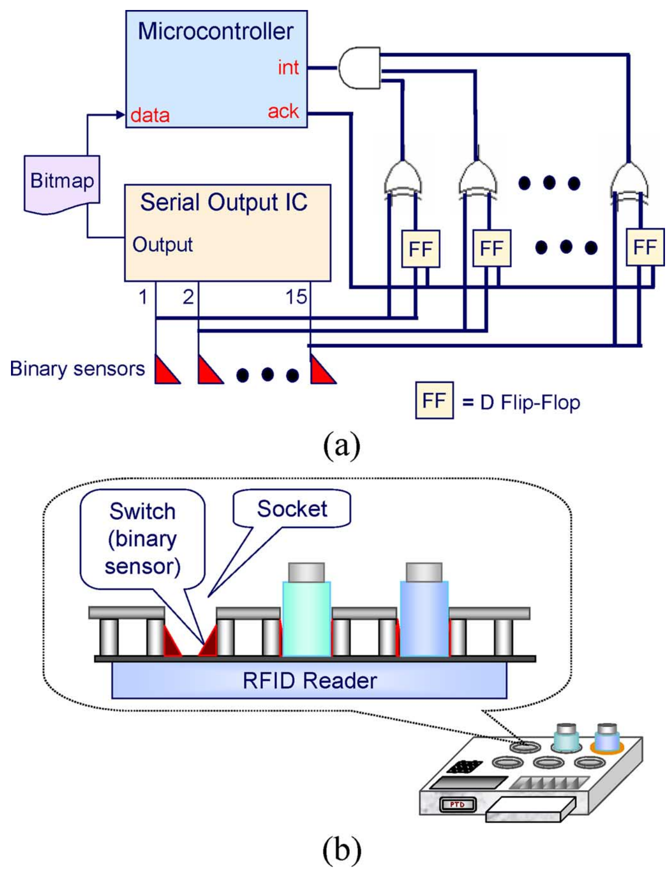


Fig. 3. Binary sensor array.

Similarly, the controller can command the RFID reader to read tags on the containers by calling Event ReadTags (char* Buffer, &Status, Timeout). When invoked, the function returns a completion event immediately while the device driver commands the RFID reader to read in nonaddressed mode. When read completes, the driver sets the completion event. The controller goes to wait for the completion event after it issues a read-tag command. When it wakes up by the event, it can determine from the returned status whether the read operation succeeded and, if the operation succeeded, the ids of tags read and returned by the reader driver.

The devices used by the dispenser to send reminders may be sophisticated or simple. The prototype uses an audio device capable of playing different tones or voice messages to indicate the urgency of the reminder. Its driver provides control functions ReminderOn (int urgency) and ReminderOff () for turning the device on and off, specifying the tone and the voice message to play when turned on. The driver of the PTD button provides no command function. It communicates with the controller via two events, one for pressing down the button and the other for releasing the button.

V. Action-Oriented Collaboration

As stated earlier, the controller and the scheduler collaborate in an action-oriented manner. Each of the collaborative entities plays one of two roles: a *decision maker* and an *action executor*. Only one entity can be the decision maker. In our smart dispenser prototype, the medication scheduler is the decision maker. The dispenser controller is the one and only action executor. By an *action*, we mean an atomic unit of work carried out by an action handler. Actions are prioritized. The handler of each action is executed as a work item by a worker thread at the priority of the action.

TABLE II
Decision-Maker API Functions

- **Void SetInformation** (InformationType, InformationData): This function allows the executor to deliver information to the DM.
  - InformationType gives the type of data structure containing the information to be delivered.
  - InformationData supplies a pointer to the data structure holding the information to be delivered.
- **Void GetInformation** (InformationType, InformationData): This function allows the caller to get information from the DM. Parameters are of the same types as those of SetInformation ( ) except that they are for data to be returned from the DM.
- **ActionDescription GetNextAction** (SystemTime CurrentTime): This function allows the caller to query the DM for actions to be performed by the caller.
  - CurrentTime provides the current system time.
  - The function returns a pointer to a structure of type ActionDescription {ActionList, NextHandShakeTime}
- **ActionDescription ActionComplete** (ActionType, ActionResult): This function allows the caller to notify the DM that the specified action is completed.
  - ActionType specifies the type of completed action.
  - ActionResult provides a pointer to result of the completed action.
- **Void EventNotify** (EventType, EventParameters) This function allows the caller to notify the DM of occurrences of an event of a specified type.
  - EventType specifies the type of event.
  - EventParameters supplies a pointer to parameters that the DM needs to decide how to handle the event.

A. Decision Maker Interface

We adopt here the variant of the action-oriented model where the executor plays a purely passive role. It is the only time keeper in the system. While it is aware of the time, it relies on the decision maker to specify the time instants for it to query for actions and the actions it is to execute at those instants. We call the nearest future time instant at which the decision maker requests the executor to query for actions the *next hand shake time* (NHST). An operation cycle of the collaborative process starts by the executor sending the decision maker the current time and the decision maker sending the executor the NHST. When given a NHST, the executor waits until the specified time to query the decision maker for action. In response, the decision maker always provides the executor with a new NHST, and may request some action(s) to be executed at that time. Thus it starts a new operation cycle.

Table II lists the basic API functions provided by the interface of the decision maker, which is referred to as the DM in the table to save space. The functions SetInformation ( ) and GetInformation ( ) allow the caller to deliver and get various types of information to and from the decision maker.

The work horse is GetNextAction (CurrentTime). By calling this function, the executor queries the decision maker for the actions to carry out, while informing the decision maker of the current time. GetNextAction ( ) returns a future next handshake

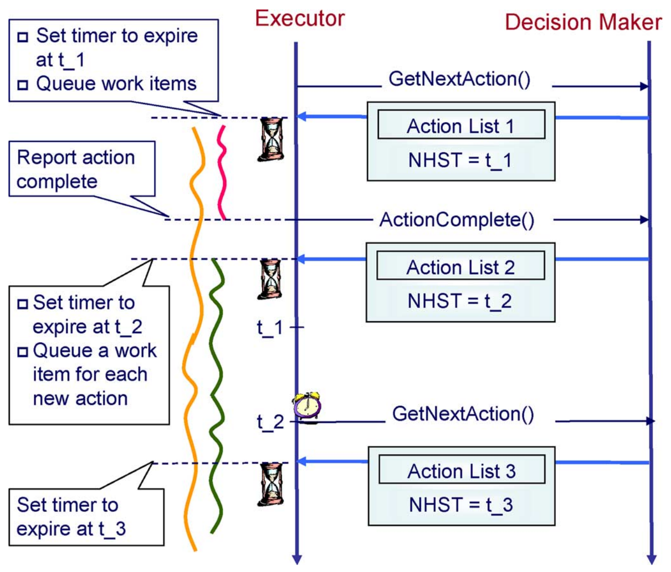


Fig. 4. Communication between executor and decision-maker.

time (NHST) and an action description. The *action description structure* contains an action list, which is NULL when the decision maker requests no action, or is the head of a list of ActionItem structures. The fields of ActionItem structure include the name of an action, a pointer to parameters to be passed to the action handler, the priority of the action and a Report flag. The flag indicates whether the result produced by the action is to be returned, and a pointer to the location for returned result.

The executor calls ActionComplete ( ) to report the completion of a specified action and to deliver the result produced by the action. The function also serves as a query for next actions as it also returns an action description and NHST.

The executor is also the only component in the system that monitors all events in the system that warrant attention. It can call the API function EventNotify() to inform the decision maker of occurrences of events that the decision maker needs to participate in handling.

B. Communication Flow

The timing diagram in Fig. 4 illustrates the protocol governing the communication between the executor and the decision maker. In the diagram, time flows downward. The exchange between the components starts from a call of GetNextAction( ) by the executor. When the function returns the value of NHST (say t_1) along with an action list, the executer first set the NHST timer to expire at that time. It then creates a work item for each action in the returned action list and queues the item for execution at the priority of the action. The wriggly lines in diagram represent threads executing the work items.

The figure shows the case where one of the actions completes before t_1, and the action result is to be returned. The executor calls ActionComplete() to report action complete and to deliver the result. The decision maker decides to request a new action and specifies a later NHST of t_2. The executor, therefore, resets the NHST timer to expire at t_2, and creates and queues a work item for each action to be executed.

Now, suppose that all the pending actions remain incomplete when the NSHT timer expires at t_2. Since t_2 is the appointed time for the executor to query for action again, the executor does so as requested. In the case shown here, the action list returned at t_2 is NULL. So, the executor resets the NHST timer to returned NHST value t_3 but queues no work item.

VI. Controller and Scheduler Design

Again, the medication scheduler is the decision maker and dispenser controller is the sole executor. This division of labor simplifies the design and implementation of both components. Having the controller as the only time keeper makes it easy to order actions in time. The design relieves the medication scheduler from the need to monitor time and external events. It now only needs to provide the API functions listed in Table II and works when its API functions are called.

A. Operational Specification

Fig. 5 gives another view of these parts. The workflow graph [27] on the top labeled "User" specifies the actions of the user. It is a part of the model that defines user behavior.

The bottom part of labeled "Dispenser Application" contains workflow graphs of the dispenser, in particular, the workflow graphs that specify the behavior of the controller and scheduler. The graphs are components of the operational (i.e., behavior) specification [28] of the dispenser, simplified to contain only parts needed for discussions in the paper.

We use the operational specification from which Fig. 5 is extracted to document the design and implementation of the dispenser. The specification can be compiled into C# code on Microsoft .NET 3.5 Workflow Foundation [29] and run in SISARL Simulation Environment [30]. We used them as executable models of the device and the user when we simulated the device and its interaction with the user for evaluation purposes. The next section will describe some of the results thus obtained.

B. Controller Software Structure

Fig. 6 shows the control structure of the prototype dispenser controller and its connection with the medication scheduler. In addition to carrying out actions requested by the scheduler, the controller also initiates actions in response to occurrence of events indicating conditions that warrant attention. It relies on an extensible library of action handler functions to carry out actions. Each action is assigned a priority by the component that requests the action. Whenever possible, executions of action handlers are scheduled preemptively according to priorities of the actions.

1) Work Queues and Worker Threads : The structure of the controller is based on a variation of the well known leader/follower pattern. During initialization, the controller (main) thread creates a pool of worker threads and several FIFO work queues of different priorities. Work items inserted into each queue are processed by worker threads at the priority of the queue. The controller thread also serves as the work item dispatcher. When

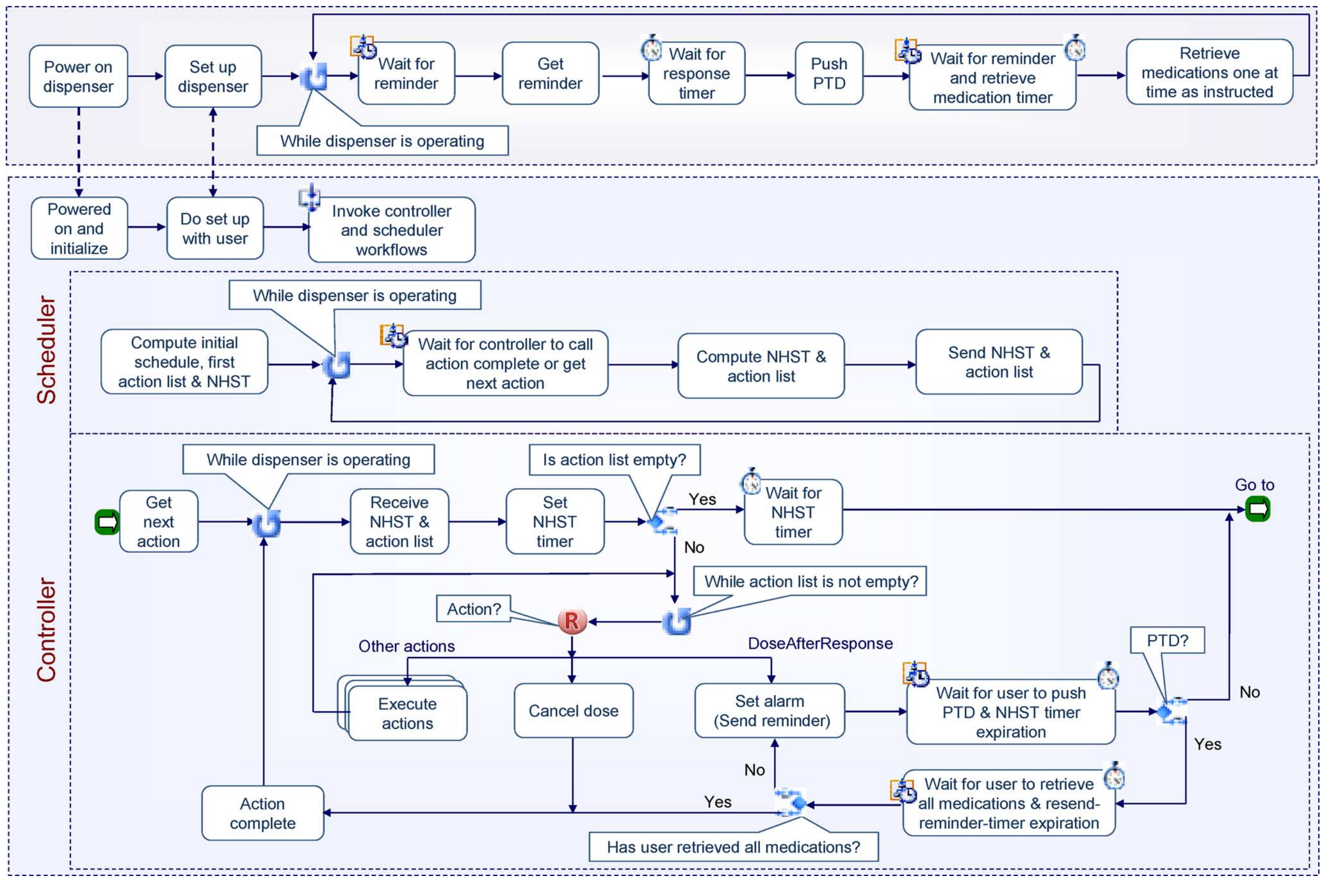

User
Power on dispenser
Set up dispenser
Wait for reminder
Get reminder
Wait for response timer
Push PTD
Wait for reminder and retrieve medication timer
Retrieve medications one at time as instructed
While dispenser is operating
Dispenser Application
Powered on and initialize
Do set up with user
Invoke controller and scheduler workflows
Scheduler
While dispenser is operating
Compute initial schedule, first action list & NHST
Wait for controller to call action complete or get next action
Compute NHST & action list
Send NHST & action list
Controller
While dispenser is operating
Is action list empty?
Get next action
Receive NHST & action list
Set NHST timer
Yes
Wait for NHST timer
Go to
No
While action list is not empty?
Action?
Other actions
DoseAfterResponse
PTD?
Execute actions
Cancel dose
Set alarm (Send reminder)
Wait for user to push PTD & NHST timer expiration
No
Yes
No
Yes
Action complete
Wait for user to retrieve all medications & resend-reminder-timer expiration
Has user retrieved all medications?

Fig. 5. Parts of operational specification of the dispenser and user actions.

it gets one or more actions from the scheduler or initiates actions on its own, it looks up the action handler functions that carry out the actions and the priorities of the actions. It wraps the pointer to each function in an instance of WorkItem data structure, along with pointers to a structure of function parameters and where result is to be returned, and sets the Report flag in WorkItem structure to the value provided by the Report field of the corresponding action item. The dispatcher then inserts the work item into one of the queues according to the priority of corresponding action. Upon finishing its dispatching duty, the controller thread returns to wait for the completion of the actions and other events.

At any time, each of the work queues is monitored by a worker thread, called the monitor thread of the queue in Fig. 6. When the thread finds work item(s) in the queue, it removes the one at the head of the queue, wakes up a worker thread in the pool to serve as the monitor thread of the queue and then calls the action handler function pointed to by the work item. When the function returns, the thread notifies the controller thread that the work has been completed and returns to wait in the thread pool. If the Report flag is set, the controller thread returns the result produced by the action handler function to the medication scheduler.

2) Event Notification: The controller uses events to notify the medication scheduler of conditions that requires attention of the scheduler, as illustrated by Fig. 6. Examples include that the PTD button is pressed. When notified of the event, the scheduler checks the existing medication schedule and makes adjustment in dose size(s) if needed.

As stated earlier, the controller is responsible for monitoring all device conditions. In addition to general conditions (e.g., power on/off), dispenser-specific conditions monitored by sensors include the ones concerning medication supplies, MSS and BSA. The sensor threads within the controller set events when the supply in some container is running low, the MSS flash disk has been plugged in and MSS file read and some container has been plugged in or removed. The events used for these purposes are MedicationInsufficient, MSSChanged, and BSAStatusChanged, respectively. The occurrences of the conditions signaled by these events may warrant that the user be alerted, the medication schedule be re-computed, and sometimes even a professional care taker be alerted, and so on. The controller calls the function EventNotify( ) in Table II to notify the scheduler whenever it cannot handle a event without the assistance of the scheduler

C. Illustrative Example

To illustrate the collaboration between the controller and the scheduler, as well as how the dispenser works to prevent serious medication administration error, we consider a simple example in which the user takes 20 mg of vitamin once daily and 10 mg of insulin every 4 hours. Without loss of generality, suppose that the dispenser is set up prior to 8:00 o'clock. According to the schedule computed by the medication scheduler immediately

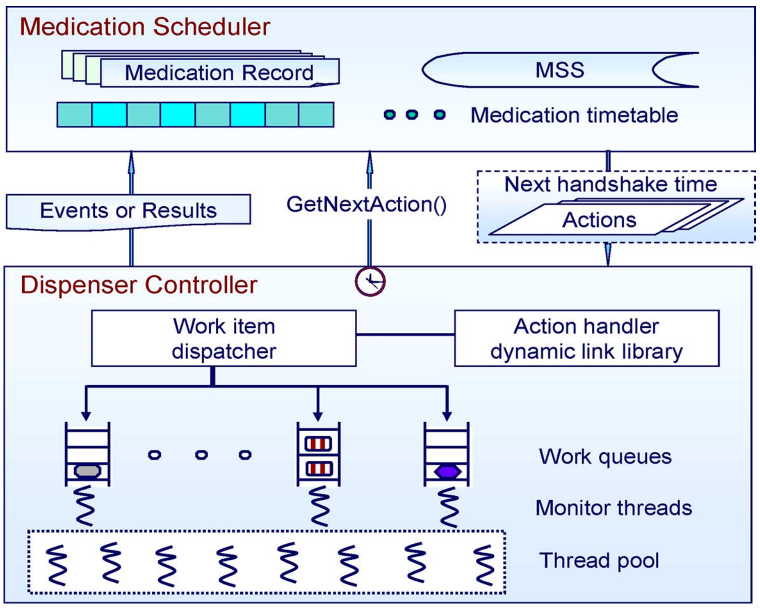


Fig. 6. Dispenser controller structure.

after set up operation completes, the user is to retrieve the daily dose of vitamin at 8:00 and 10 mg doses of insulin at 9:00, 13:00, 17:00, and so on.

While Vitamin can be skipped with little or no consequence, doses of insulin cannot be skipped. Its direction says: First, when the user is tardy for more than 4 hours, the pending dose is cancelled, and a double-size dose of 20 mg is scheduled at the next dose time. Second, contact the user's doctor if the user has not taken insulin for 10 hours or more. The relaxation from a dose every 4 hours schedule allowed by the first rule permits the user to have 8 hours of uninterrupted sleep. The second rule defines a noncompliance event and ways to handle it.

Fig. 7 shows the interaction between the controller and the scheduler. Again, time flows downwards and is not drawn in scale. The wiggly lines at the left edge represent running worker threads that turn the reminder on or off, monitor user response, help the user retrieve a dose, and handle the noncompliance event. To save space, we put a "*" in front of an action name to indicate that the scheduler wants the result of the action returned. The operation starts from the controller making a GetNextAction( ) call. In response, the scheduler asks to be queried at 8:00 o'clock, time for vitamin. The controller sets the NHST timer to expire at 8:00 and goes to wait.

- At 8:00 when the NHST timer expires, the controller queries the scheduler for action, telling the scheduler that the current time is 8:00 o'clock. The $\mathtt{NHST} = 9:00$ returned by the scheduler is the next dose time for insulin. The action list returned by the scheduler specifies three actions: The SetAlarm (on, 0) action turns on the reminder. Since vitamin may be skipped, the scheduler sets the persistence parameter to 0, telling the controller that the reminder can be turned off automatically after a brief interval of time. The second and third actions in the list tell the controller to be ready to help the user retrieve 20 mg of vitamin when the user pushes the PTD button. After queuing work items for the actions, the controller sets the NHST timer to expire at 9:00 and returns to wait for the timer.
- Suppose the user responds to the reminder and pushes the PTD button at 8:10 while the reminder is still on. The controller helps the user retrieve from the vitamin container a 20 mg tablet and then calls ActionComplete( ) to return the actual dose time of 8:10. As this function is also a query for the next action, the scheduler returns two actions: They are to turn off the reminder and stop monitoring the PTD button. After these actions are dispatched, the controller goes to wait for the NHST timer to expire at 9:00.
- Suppose that all went well prior to 13:00: The user promptly retrieved the 9:00 dose of insulin. When the controller calls GetNextAction() at 13:00, the scheduler requests that the reminder be turned on, this time persistently until the scheduler requests for it to be turned off. Then the controller is to be ready to help the user retrieve a 10 mg dose of insulin when the user pushes the PTD button. After dispatching the work items for these actions, the controller sets the NHST timer to expire at 17:00 and goes to wait.
- At 17:00, the user still has not responded. When the controller calls GetNextAction(), the fact that the controller has not reported the completion of the DoseAfterResponse action it requested at 13:00 tells the scheduler that the dose of insulin scheduled at that time is still pending. Moreover 4 hours has elapsed. The schedule needs to be adjusted. Hence, the scheduler requests that the pending dose be cancelled, while it adjusts the schedule according to MSS. When the controller reports the completion of CancelDose, the scheduler requests that a 20 mg dose is to be given to the user when the user responds. The reminder is still on, and the controller is still paying close attention to the PTD button. The value of NHST returned by the scheduler is 19:00.

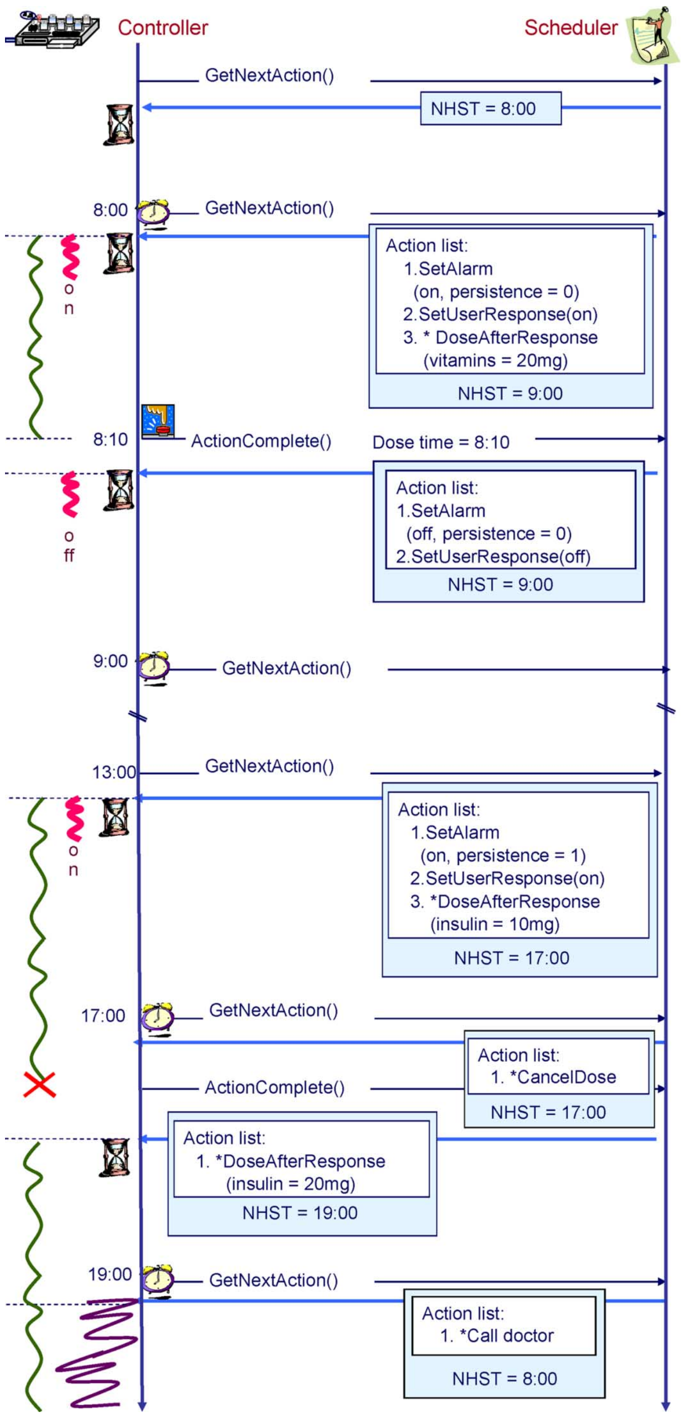


Fig. 7. An illustrative example.

- By 19:00, 10 hours have elapsed since the user took the latest dose of insulin. The user still has not come to push the PTD button by 19:00. So, the scheduler requests that the controller calls the designated care taker to report the noncompliance event. The value of NHST is 8:00, the time for dose of vitamin for the next day. In the meantime the 20 mg dose of insulin is still pending.

VII. Evaluation

We have evaluated the dispenser using both formal method and simulation approaches. Clearly, a critical requirement of the dispenser is that it works correctly. This means that it carries out all medication administration operations according to the rules and constraints defined by the user's MSS and it handles all exceptions (and faults) dependably.

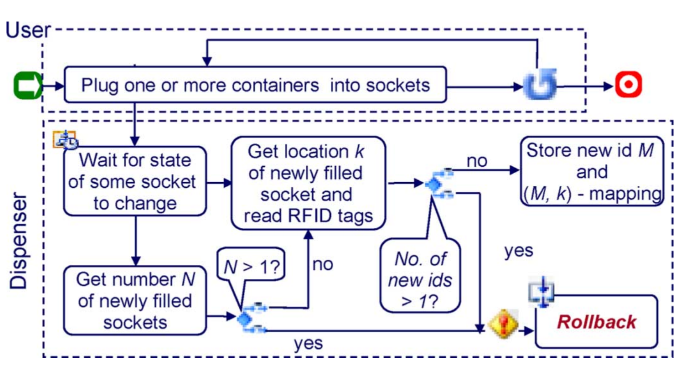
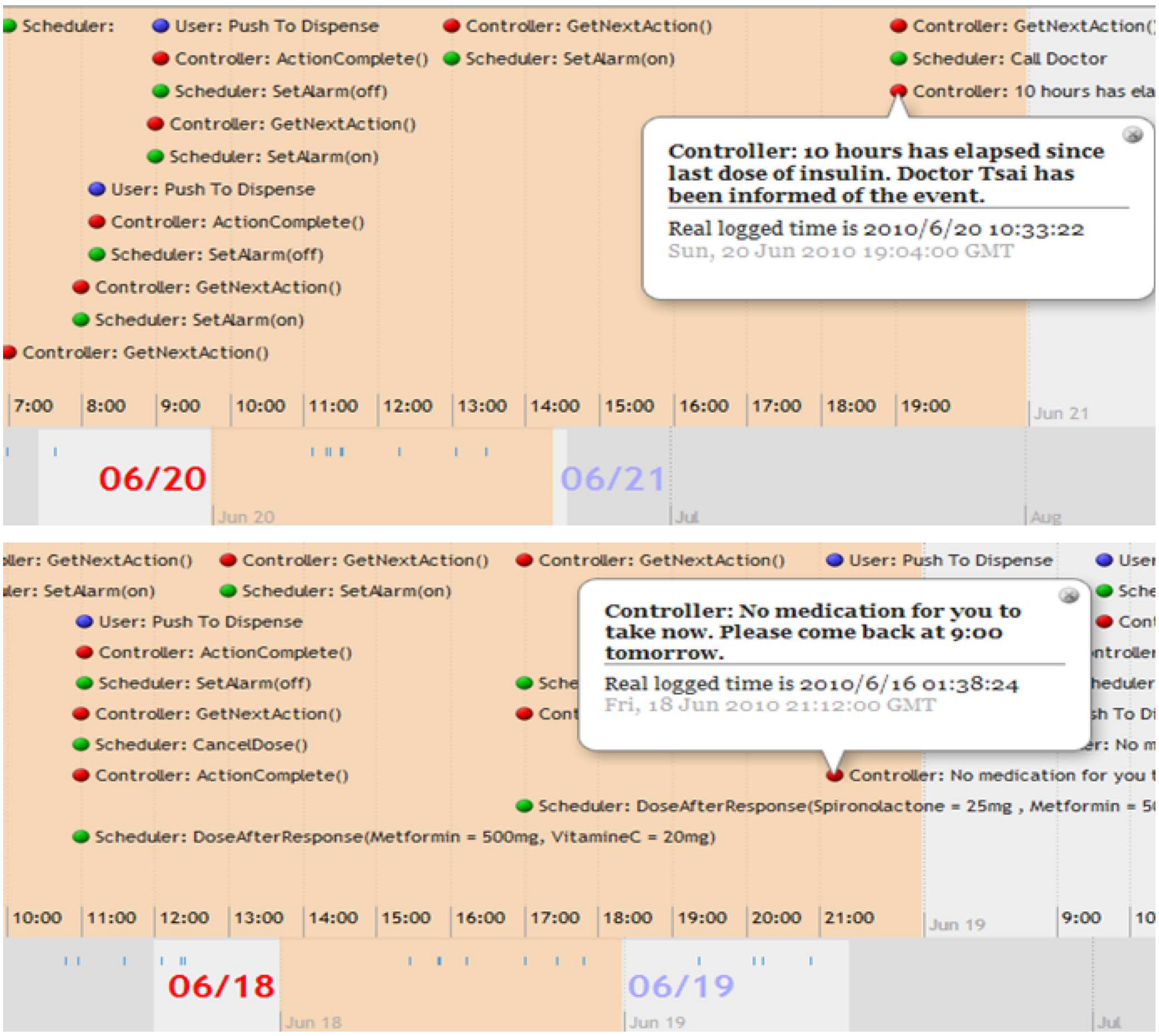

User
Plug one or more containers into sockets
Dispenser
Wait for state of some socket to change
Get location k of newly filled socket and read RFID tags
no
Store new id M and (M, k) - mapping
yes
Get number N of newly filled sockets
N > 1?
no
No. of new ids > 1?
yes
Rollback

Fig. 8. Operational specification of set up operation.

Exceptions can arise from device malfunctions as well as incorrect use and misuse by the user. We have been focusing on the latter: Our goal is to identify and correct design flaws that allow incorrect user actions to be undetected, delaying time critical recovery actions. As an example, the design of dispenser set up operation is defined by the part of operational specification shown in Fig. 8. The assumption underlying the design is that the containers are plugged into empty sockets one at the time. It treats conditions indicating possible violation of this assumption as exceptions and handles them by prompting the user to rollback (i.e., unplugs the containers in the sockets involved and plugs them back one at a time). Here, the correctness criterion is that the dispenser correctly locates the container of every medication in user's MSS after set up. We can prove the correctness of this design using model checking tools such as RED [31]. Indeed, we can use this approach to verify the designs of handlers of most exceptions for which the times of occurrences are not important.

We rely on simulation to catch design flaws that can cause errors in medication administration, including mistakes in medication schedules and failures to raise and handle noncompliance events. Simulation experiments were done in the SISARL Simulation Environment (SSE) [29]. For each simulation experiment, we provide the tool with three kinds of input. This first is the operational specification of the device. Again, the specification defines the device behavior, user actions and device-user interactions. Parts of the specification of the dispenser are illustrated by Figs. 5 and 8.

Second kind of input to a simulation experiment includes user models. In general, SSE enables us to use all variants of the GOMS model to capture user behavior at appropriate levels of detail [31]. For simulating the dispenser, it suffices to characterize the user by a set of parameters that allow us to compute the lengths of time the user take to complete elementary steps in user activities in the user workflow. The most important parameter is the probability distribution of the user response time (i.e., the time between when a reminder is sent to the time when the user comes to push PTD button). During each simulation run, timeout length of the response timer is set to a sample value from this distribution.



Scheduler:
User: Push To Dispense
Controller: ActionComplete()
Scheduler: SetAlarm(off)
Controller: GetNextAction()
Scheduler: SetAlarm(on)
User: Push To Dispense
Controller: ActionComplete()
Scheduler: SetAlarm(off)
Controller: GetNextAction()
Scheduler: SetAlarm(on)
Controller: GetNextAction()
Controller: GetNextAction()
Scheduler: SetAlarm(on)
Controller: GetNextAction(
Scheduler: Call Doctor
Controller: 10 hours has ela

Controller: 10 hours has elapsed since last dose of insulin. Doctor Tsai has been informed of the event.
Real logged time is 2010/6/20 10:33:22
Sun, 20 Jun 2010 19:04:00 GMT

ller: GetNextAction()
ler: SetAlarm(on)
User: Push To Dispense
Controller: ActionComplete()
Scheduler: SetAlarm(off)
Controller: GetNextAction()
Scheduler: CancelDose()
Controller: ActionComplete()
Scheduler: DoseAfterResponse(Metformin = 500mg, VitamineC = 20mg)
Controller: GetNextAction()
Scheduler: SetAlarm(on)
Controller: GetNextAction()
Sche
Cont
Scheduler: DoseAfterResponse(Spironolactone = 25mg , Metformin = 5
User: Push To Dispense
Controller: No medication for you t
User
Sche
Cont
ntroller
heduler
sh To Di
er: No m

Controller: No medication for you to take now. Please come back at 9:00 tomorrow.
Real logged time is 2010/6/16 01:38:24
Fri, 18 Jun 2010 21:12:00 GMT

7:00 8:00 9:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 Jun 21
06/20 Jun 20 06/21 Jul Aug
10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 Jun 19 9:00 10
06/18 Jun 18 06/19 Jun 19 Jul

Fig. 9. Examples of simulation data.

The third kind of input consists of number, kinds and constraint parameters of medications in the user's MSS. We use many real-life prescriptions as basis to generate sample MSS. We also use synthetic MSS generated randomly from specified probability distributions as described in [25].

During simulation, the tool captures events in interactions of the user, controller and scheduler and logs them in a database. As example, the scheduler workflow generates an error event when it cannot determine next action or NHST. In addition, we add conditional rules in the workflows. The rules enable us to check whether the controller instructed the user to retrieve the correct dose of each medication at each dose time. We check for correctness by processing and analyzing the logged events to find error events. We also can detect errors by studying timeline displays of the event traces.

To illustrate, Fig. 9 shows fragments of event traces captured in two simulation experiments. We say that the user is prompt if he/she responds to reminder soon enough that there is no need to adjust the medication schedule. Otherwise he/she is tardy by a random amount of time. The simulation traces are both for a user who is tardy about 30% of the time. Each dot on the timeline represents an event. We can view detailed information and the actual logged time of the event by clicking the dot.

Specifically, the top trace in Fig. 9 is from a simulation run where the user is on the medications described in Section VI(C). The trace shows that the dispenser correctly handles the non-compliance event described in the example. The bottom event trace is from a simulation run where is the user is on the three medications shown in Fig. 10. The segment shows an event raised by the controller when the user pushes the PTD button at times when no medication is due.

VIII. Summary and Future Work

Previous sections described the design and operations of a smart medication dispenser. Except for the set up and dose retrieval operations, the dispenser is fully automatic. The work

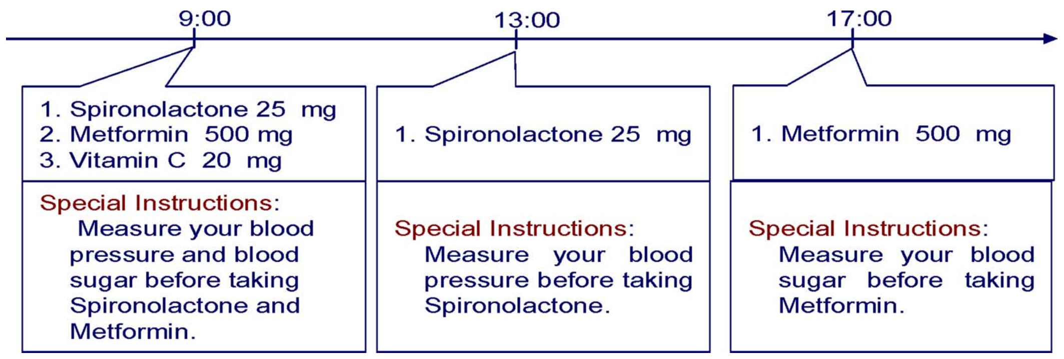

9:00
13:00
17:00
1. Spironolactone 25 mg
2. Metformin 500 mg
3. Vitamin C 20 mg
Special Instructions: Measure your blood pressure and blood sugar before taking Spironolactone and Metformin.
1. Spironolactone 25 mg
Special Instructions: Measure your blood pressure before taking Spironolactone.
1. Metformin 500 mg
Special Instructions: Measure your blood sugar before taking Metformin.

Fig. 10. A medication schedule used for simulation.

of medication administration is done collaboratively by the dispenser controller and the medication scheduler in an action oriented manner. By replacing the decision maker and action handlers, one can build a different device using more or less the same action executor. Similar, one can easily enhance and configure the medication dispenser by adding new and enhance action handlers into the action handler library.

The dispenser controller and the medication scheduler in the current prototype run on the same computer. An alternative design is to have the controller run on the local embedded machine, but to have the scheduler runs on a networked server. In that case, we can link multiple dispensers to a single scheduler server through the network and have the scheduler compute medication timetables for multiple persons and request corresponding actions to be carried out by multiple dispensers. This design is suitable for dispensers in hospital and clinics, but factors such as high communication overheads and lower network availability may rule out its use for dispensers serving individuals at home or work.

Many functions can be added to the dispenser to improve its user friendliness and effectiveness. The current prototype does not support the capture and use of user preferences. The medication scheduling algorithms need to be modified to take into account the soft constraints defined by the user preference parameters. The compliance monitor in the current prototype is designed to perform basic noncompliance notification (e.g., call a care taker). In addition to enabling it to send noncompliance notifications using multiple media, we also want the dispenser to record the actual sizes and times of individual doses of all medications taken by the user, and thus generate a local medication record for the user.

Finally, we have validated and evaluated the dispenser to the extent needed to demonstrate that its design is sound. Much work on testing, experimentation and trial use (by actual users) of the proof-of-concept prototype remains to done to turn the prototype into a mature product.

ACKNOWLEDGMENT

The authors wish to thank members of SISARL medical advisory board, listed at http://sisarl.org, and Dr. D. H. Burkhardt for their critiques and suggestions.

REFERENCES

[1] "Preventing Medication Errors," Report Brief Inst. Medicine Nat. Academies [Online]. Available: http://www.iom.edu/Object.File/Master/35/943/medication%20errors%20new.pdf

[2] *Drug Safety and Availability: Medication Errors*, , Jun. 18, 2009 [Online]. Available: http://www.fda.gov/Drugs/DrugSafety/MedicationErrors/default.htm

[3] P. J. Veacez, "An individual based framework for a study on medical error," *Int. J. Qual. Health Care*, vol. 18, no. 4, May 2006.

[4] M. Lisby, L. P. Nielsen, and J. Mainz, "Errors in the medication process: Frequency, type, and potential clinical consequences," *Int. J. Qual. Health Care*, vol. 17, no. 1, 2005.

[5] G. J. Kuperman *et al.*, "Medication related clinical decision support in computerized provider order entry systems: A Review," *J. Amer. Med. Inform. Assoc.*, 2007.

[6] *Health Information Systems*, , 2005 [Online]. Available: http://www.hhs.gov/healthit/ahic.html

[7] D. M. Cutler, N. E. Feldman, and J. R. Horwitz, "U.S. adoption of computerized physician order entry systems," *Health Affairs*, vol. 24, no. 6, 2005.

[8] R. L. Davis, "Computerized physician order entry systems: The coming of age for outpatient medicine," *PLoS Med.*, 2005.

[9] B. Koppel *et al.*, "Role of computerized physician order entry systems in facilitating medication errors," *J. AMA*, vol. 293, no. 10, 2005.

[10] W. J. King *et al.*, "The effect of computerized physician order entry on medication errors and adverse drug events in pediatric inpatients," *Pediatrics*, Sep. 2003.

[11] J. Smaling and M. A. Holt, "Integration and automation transform medication administration safety," *Health Care Manag. Technol.*, Apr. 2005.

[12] P. Bonnabry, "Information technologies for the prevention of medication errors," *Eur. Pharmacotherapy*, 2003.

[13] S. C. Dursco, "Technological advances in improving medication adherence in the elderly," *Ann. Long-Term Care: Clinical Care and Aging*, vol. 9, no. 4, 2001.

[14] Drug Information PDRHealth [Online]. Available: http://www.pdrhealth.com/drug_info/

[15] H. C. Yeh, P. C. Hsiu, C. S. Shih, P. H. Tsai, and J. W. S. Liu, "APAMAT: A prescription algebra for medication authoring tool," in *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, Oct. 2006.

[16] Pill Dispenser E-pill [Online]. Available: http://www.epill.com/dispenser.html and MD 2: http://www.epill.com/md2.html

[17] *Pill Boxes*, [Online]. Available: http://www.dynamic-living.com/automated_medication_dispenser.htm

[18] *My Pill Box*, [Online]. Available: http://www.mypillbox.org/mypillbox.php

[19] *Rx Showcase*, [Online]. Available: http://www.rxinsider.com/prescription_dispensing_automation.htm

[20] M. Governo, V. Riva, P. Fiorini, and C. Nugent, "MEDICATE teleassistance system," in *11th Int. Conf. Advance Robotics*, Jun. 2003.

[21] M. D. Murray, "Automated medication dispensing devices," in *Making Health Care Safer: A Critical Analysis of Patient Safety*. : Agent for Healthcare Research and Quality, 2001, vol. 01-E58, ch. 11.

[22] D. Wan, "Magic medicaine cabinet: A situated portal for consumer healthcare," in *Proc. First Int. Symp. Handheld and Ubiquitous Computing (HUC '99)*, Sep. 1999.

[23] M.-Y. Wang, J. K. Zao, P. H. Tsai, and J. W. S. Liu, "Wedjat: A mobile phone based medication reminder and monitor," in *IEEE Int. Conf. on BioInformatics and BioEngineering*, Jun. 2009.

[24] P. H. Tsai, H. C. Yeh, C. Y. Yu, P. C. Hsiu, C. S. Shih, and J. W. S. Liu, "Compliance enforcement of temporal and dosage constraints," in *Proc. IEEE Real-Time Systems Symp.*, Dec. 2006.

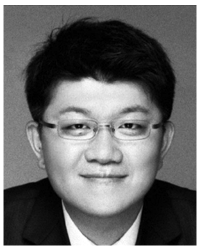
[25] P. H. Tsai, C. S. Shih, and J. W. S. Liu, "Algorithms for scheduling multiple interacting medications," *Found. Comput. Decision Sci.*, vol. 34, no. 4, 2009.

[26] P. C. Hsiu, H. C. Yeh, P. H. Tsai, C. S. Shih, D. H. Burkhardt, T. W. Kuo, J. W. S. Liu, and T. Y. Huang, A General Model for Medication Scheduling Inst. Inform. Sci., Academia Sinica, Taiwan, 2005, Technical Report TR-IIS-05-008.
[27] B. Bukovics*, Pro WF: Windows Workflow Foundation*. New York: Apress, 2009, vol. .Net 4.0.
[28] T. Y. Chen, P. H. Tsai, T. S. Chou, C. S. Shih, T. W. Kuo, and J. W. S. Liu, "Component model and architecture of smart devices for the elderly," in *Proc. 7th Working IEEE/IFIP Conf. Software Architecture*, 2008.
[29] T. Y. Chen, C. H. Chen, C. S. Shih, and J. W. S. Liu, "A simulation environment for the development of smart devices for the elderly," in *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, 2008.
[30] F. Wang, "RED: Model-checker for timed automata with clock-restriction diagram," in *Workshop on Real-Time Tools*, Copenhagen, Denmark, Aug. 20, 2001.
[31] S. K. Card *et al., The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum, 1983.

Pei-Hsuan Tsai received the M.Eng. degree in computer science from Cornell University, Ithaca, NY, in 2004. She is currently pursuing the Ph.D. degree in computer science at National Tsing Hua University, Taiwan and expects to complete her Ph.D. study in 2010. She plans to join Academia Sinica as a postdoc after her graduation. Her research interests are embedded systems, hospital automation, scheduling algorithms and UI architecture and design.

Tsung-Yen Chen received the M.S. degree in computer science in 2004 from the National Tsing Hua University, Taiwan, where he is currently pursuing the Ph.D. degree in computer science. His research interests are embedded system, human-computer interactions, and system modeling and evaluation.

Chi-Ren Yu received the M.S. degree in computer science in 2008 from the Department of Computer Science from National Tsing Hua University, Taiwan. His research interests are smart devices, embedded software and systems. He is currently an engineer in the SOANDSO department (or group) at Mediatek Company.



Chi-Sheng Shih (M'03) received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 2003.

He has been an Associate Professor with the Department of computer science and information engineering, National Taiwan University, Taipei, since August 2008. His main research interests are real-time systems and database systems. Specifically, his main research interests focus on object-relational database query optimization, real-time operating systems, real-time scheduling theory, embedded software, and software/hardware co-design for system-on-a-chip.

Dr. Shih received the Best Student Paper Award from the IEEE Real-Time System Symposium in 2004 and Best Paper Award from the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications in 2005.

Jane W. S. Liu (F'95) received the B.S. degree in electrical engineering from Cleveland State University, Cleveland, OH, and the Sc.D. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge.

She was a Software Architect in the OS Base Core Technology group of Microsoft Corporation from 2000 to 2004 and was a faculty member of Computer Science Department at the University of Illinois at Urbana-Champaign from 1973 to 2000. Her research interests are real-time and embedded systems and distributed systems. Her recent research focuses on technologies for building personal and home automation and assistive devices and services. In addition to journal and conference publications, she has also published two books: *Real-Time Systems* (Prentice-Hall, 2000) and *Linear Systems Analysis* (Prentice-Hall, 1975)..

Dr. Liu was the editor-in-chief of IEEE TRANSACTIONS ON COMPUTERS from 1996 to 1999 and is a member of editorial board of *Real-Time Systems*. She won the Outstanding Technical Achievement Award of IEEE Computer Society, Technical Committee on Real-Time Systems, in 2005, the Information Science Honorary Medal from Taiwan Institute of Information and Computing Machinery in 2008 and the Taiwan Linux Consortium Golden Penguin Award for special contributions in 2009.