中央研究院
資訊科學研究所
Institute of Information Science, Academia Sinica • Taipei, Taiwan, ROC

TR-IIS-06-006

Compliance Enforcement of Temporal and Dosage Constraints

P.H. Tsai, H.C. Yeh, C.Y. Yu, P.C. Hsiu, C.S. Shih, J.W.S. Liu

May 29, 2006 || Technical Report No. TR-IIS-06-006

http://www.iis.sinica.edu.tw/LIB/TechReport/tr2006/tr06.html

Compliance Enforcement of Temporal and Dosage Constraints

P. H. Tsai, H. C. Yeh, C. Y. Yu, P. C. Hsiu, C. S. Shih, *Member, IEEE*, J. W. S. Liu, *Fellow, IEEE*

***Abstract*—Medication dispensers treated in this paper are designed to help improve compliance by users who live at homes and take medications over long periods of time. The paper first presents an overview of medication specifications that define constraints for dispensers and dispenser components that administer medications as specified. When given a specification and constraints defined by it, the dispenser scheduler checks for consistency and feasibility of constraints and schedules medications to meet the constraints. Several basic algorithms needed for these purposes are described and evaluated.**

I. Introduction

These days, one can find on-line and in specialty stores numerous devices and services designed to ease the effort and improve the chance in medication compliance. They include dumb pillboxes and programmable medicine dispensers (e.g., [1, 2]), as well as websites that help the user to generate medication schedules (e.g., [3]). Modern drugs can do wonders in controlling deceases and maintaining health, but only if the user follows the prescribed directions. Unfortunately, non-compliance is far too common and severe [4, 5], especially for elderly and chronically ill individuals: Such an individual may still live at home, on several prescribed and over-the-counter (OTC) medications at a time, and have 10 or more different prescriptions each year for many years, even decades. Existing devices and services are not ideal in many ways: They typically require manual handling of the medications, and the schedules they support are rigid. For long term users, it is essential that medication schedules are as flexible as possible. The dispenser must be tolerant to user tardiness since tardiness is unavoidable.

This paper describes architecture of smart medication dispensers that are designed for flexibility and tolerance and basic algorithms that a dispenser scheduler can use for compliance enforcement. By a *smart medication dispenser* (or *dispenser* for short), we mean specifically a device for use by a naive user at home without close professional supervision. During normal operation, the dispenser schedules the user's medications, reminds the user at times when medications should be taken, controls the dosages dispensed each time, and dynamically readjust the medication schedule to stay compliant when the user is tardy. The dispenser provides appropriate warnings when it becomes impossible to stay compliant.

A requirement for proper usage is that all (prescription or OTC) medications taken by the user are managed by a single dispenser. The user is provided by user's pharmacist with a machine readable *Medication Schedule Specification* (*MSS*) either via Internet or a portable storage device each time the user acquires medication supplies. When loaded into the dispenser, the specification defines for the dispenser nominal temporal and dosage constraints that should be satisfied by schedules used for dispensing the medications under its care. In addition, the specification provides hard limits; they are criteria for compliance monitoring and enforcement.

The models underlying medication schedule specifications [6, 7] resemble well-known real-time work-load models (e.g., [8-15]) in many respects. The resemblance is intentional; we want to apply established real-time systems principles and existing techniques to medication scheduling wherever appropriate. We will discuss relationships between related models and algorithms in later sections where MSS and associated algorithms are described.

Our assumption is that some professional(s) has verified for each user the safety and effectiveness of the user's medications and the correctness of their directions. Hence, many difficult problems on medications addressed by medication consultation projects (e.g., [16]) are out of scope for us. On the other hand, their treatment to medication scheduling lacks the depth and rigor we need.

Following this introduction, Section II describes what and how requirements and constraints are defined in MSS. Section III describes dispenser architecture and operations. Section IV presents consistency and feasibility conditions. Section V and VI describe dosage selection and scheduling algorithms. Section VII is a summary.

II. Medication Schedule Specification

Throughout the paper, by medications we mean both prescription drugs and OTC drugs and supplements. Except for where distinction is necessary, we also refer to food as a medication. The dispenser does not handle food, but must schedule meals and snacks and send remembers for them when some of the user's medications interfere with food.

A. Usage Assumptions

We confine our attention to dispensers for individual users

P. H. Tsai and C. Y. Yu are with Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. Their email addresses are peipei@eos.cs.nthu.edu.tw and u910224@oz.nthu.edu.tw.

H. C. Yeh, P. C. Hsiu, and C. S. Shih are affiliated with Department of Computer Science and Information Science, National Taiwan University, Taipei, Taiwan. Their email addresses are {r94922048, r91004, cshih}@csie.nthu.edu.tw.

J. W. S. Liu is with Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan. Her email address is janeliu@iis.sinica.edu.tw.

who live at home without assistance and are on multiple medications, some of which are taken on long term basis. Again, we assume that all medications taken by a user are managed by a dispenser. A likely scenario is that the user is given a dispenser by the user's pharmacy, hospital or medical clinic, with expense covered in part by health insurance as a reward for improved compliance.

Smart medication dispensers are end-user devices in a tool chain for medication management and delivery. Tools at high end of the chain include various manual and computerized physician order entry systems (e.g., [17-20]). Using them together with on-line drug libraries and health information systems (e.g., [21, 22]), medical and health-care professionals can in principle verify automatically or interactively that the user can safely and effectively take the medications ordered and directed in each prescription. A requirement for correct usage of dispensers is that this verification has been done.

Another important assumption is that the pharmacist is provided with access to all prescriptions of the user, as well as information on OTC medications the user takes at the time. We are building incrementally a *prescription authoring tool* [23] to aid the pharmacist: Each time when the user comes to fill a new prescription or to purchase some OTC drug, the pharmacist uses the tool to extract user specific directions from the user's prescriptions; put the extracted directions in a unified representation; and check for conflicts while integrating user specific directions with general directions the tool extracts from drug libraries. By *conflicts*, we mean inconsistencies and flaws in directions that the tool cannot resolve automatically. If tool finds no conflict during the integration process, it translates the merged directions into a MSS for the user's dispenser. Otherwise, it sends the information on the conflicts to the responsible order entry systems and the physicians, requests conflict resolution, and repeats all the steps when the conflicts are resolved.

B. Direction Parameters

The MSS generated by the authoring tool consists of two parts: direction of each medication when taken alone and changes in directions and additional constraints when some of the user's medications interact. Fig. 1 gives a partial list of parameters that define the direction of a medication. Other parameters and further details can be found in [6].

The name M of the medication identifies the medication and provides the dispenser with information on its physical characteristics. We consider here only medications that are dispensed in discrete units. The granularity g specifies the minimum size of a dose. A granule of a medication may be a tablet or caplet, some number of milligrams (mg) or cubic centimeters (cc), etc. So, a dose of size 1 means different amounts for different medications.

We will make frequent reference to the parameters starting from $[T_{min}, T_{max}]$: The *minimum duration* T_{min} and *maximum duration* T_{max} bound the length of time over which the medication is to be administered. Normal operations of the dispenser are guided by the parameters listed in the 4 lines below duration. The dispenser treats the constraints defined by these parameters as *firm constraints*: The medication schedules it uses meet the constraints, but user tardiness may lead to the violation of some of them and trigger the re-computation of the schedule in effort to stay compliant.

- M: Name of the medication
- g: Granularity
- $[T_{min}, T_{max}]$: Minimum and maximum durations
- $[d_{min}, d_{max}]$: Nominal minimum and maximum dose sizes
- $[s_{min}, s_{max}]$: Nominal minimum and maximum separations
- (B, R): Maximum intake over a specified time interval given by budget B and replenishment delay R
- (L, P): Minimum intake over a specified time interval given by lower bound L and interval length P
- $[D_{min}, D_{max}]$: Absolute minimum and maximum dose sizes
- $[S_{min}, S_{max}]$: Absolute minimum and maximum separations
- Non-compliance event types and corresponding actions.

Fig. 1 Direction parameters

Dose size is the number of granules taken in one dose. *Separation* is the length of time between consecutive doses. *Dosage* means the combination of dose size and separation. The dispenser normally provides the user with doses of sizes in the range $[d_{min}, d_{max}]$, delimited by the *nominal minimum dose size* d_{min} and *nominal maximum dose size* d_{max}, with separations in the range $[s_{min}, s_{max}]$, delimited by the *nominal minimum separation* s_{min} and the *nominal maximum separation* s_{max} of the medication. We use hour as unit of time throughout the paper. The actual time granularity of dispensers is 10 – 100 milliseconds, however.

The *maximum intake* (B, R) requires that no more than B granules of M are taken in any time interval of length R. B stands for *budget*, and R stands for *replenishment delay*. The current budget is B initially. When a dose of size d is dispensed at time t, d granules of budget are consumed, and the d-granule chunk is replenished at $t + R$. At the time of any dose, the dose size can be at most equal to the current budget. This way of enforcing the maximum intake (B, R) constraint is similar to how a sporadic server [9, 10] limits processor bandwidth consumption of aperiodic tasks.

For some medications (e.g., antibiotics), it is important that a certain amount of the medication is at work at all times. Such a medication typically has a minimum intake constraint (L, P). It requires that the total size of doses within any interval of length P to be at least equal to the lower bound L.

As illustrative examples, we quote below the directions of a pain killer and an antibiotic.

> ***Example 1***: "1 gel caplet every 4 to 6 hours …. If pain or fever does not respond to 1 caplet, 2 caplets may be used but do not exceed 6 gel caplets in 24 hours unless directed by a doctor. The smallest effective dose should be used."
>
> ***Example 2*:** "Take 250 to 500 milligrams (mg) every eight hours. Keep taking this medicine for at least ten days …. It is best to take the doses at evenly spaced times … on an empty stomach (either 1 hour before or 2 hours after meals)."

The nominal dose size and separation of the pain killer have ranges [1, 2] and [4, 6], respectively. It has a maximum intake of (6, 24) to safeguard against over dose, but has no minimum intake. The granularity of the antibiotic is 125 mg. Its

direction allows a large dose-size range [2, 4], but suggests a narrow separation range [8, 8]. Its intakes are $(B, R) = (12, 24)$ and $(L, P) = (6, 24)$. These constraints provide a guideline for deviation from the rigid 8-hour day and night schedule.

The antibiotic in Example 2 has a minimum duration of 240 hours. This is a hard constraint, as are constraints defined by absolute dose size and separation. The dispenser treats a violation of a *hard constraint* as a non-compliance event that warrants a specified action. As shown in Fig. 1, MSS includes medication-specific rules governing how the dispenser is to define and respond to non-compliance events. This issue will be addressed in a later paper.

The *absolute minimum dose size* D_{min} and *absolute maximum dose size* D_{max} are the smallest and the largest dose sizes, respectively, amongst all values that are explicitly or implicitly specified by statements in the direction. Similarly, *absolute minimum separation* S_{min} and *absolute maximum separation* S_{max} are the smallest and largest separations, respectively, specified by direction statements. In Example 1, the absolute maximum dose size is also 2. Because of the last statement in the direction, the absolute minimum doze size is 0, and the absolute maximum separation is infinity. Directions of many medications allow large ranges of absolution separation, while suggesting no variation in nominal dose sizes and separation. An example is a brand of digoxin [11] for control of some heart conditions. Its instruction suggests taking a dose a day, at the same time each day. On how to handle a missed dose, the instruction says "If you remember within 12 hours, take it immediately. If you remember later, skip the dose you missed and go back to your regular schedule". For this medication, the nominal separation range is [24, 24], and the absolute separation range is [12, 36].

C. Interaction Pairs

The MSS captures the information on direction changes due to interactions between medications using interaction pairs. There is an *interaction pair* $I(M, N)$ for each pair of medications M and N that may interact to a degree as to require modification in how the medications are to be administered. The *attributes of the interaction pair* contains three components: direction change lists, inter-medication separations, and precedence constraints.

Some direction parameters for individual medications in an interaction pair may need to be modified. Consider Fosamax [11] for treatment and prevention of brittle bone decease as an example. When taken alone, $[d_{min}, d_{max}] = [1, 7]$, $[s_{min}, s_{max}] = [20, 168]$, and $(B, R) = (7, 168)$. However, when the user is also taking aspirin, only dose size 1 with separation in the range [20, 24] is allowed in order to minimize the chance of stomach upset. The MMS captures such changes in change lists $C(M)$ and $C(N)$ of each medications in the interaction pair $I(M, N)$. While user is taking both medications, the parameters given by the change lists replace the direction parameters of the individual medications.

The separation attribute of $I(M, N)$ includes *minimum separation* $\sigma_{min}(M, N)$ between the medications: An earlier dose of M (N) must be separated from a later dose of N (M), by at least $\sigma_{min}(M, N)$ ($\sigma_{min}(N, M)$). $\sigma_{min}(M, N)$ may not equal to $\sigma_{min}(N, M)$. In Example 2, $\sigma_{min}(Antibiotic, Food) = 0.5$ and $\sigma_{min}(Food, Antibiotic) = 1$. Some medications may be constrained to be taken sufficiently close together or taken together with food. This requirement is expressed in terms of *maximum separation,* which can be defined similarly [6].

Finally, interaction of medications may also lead to precedence constraints that dictate the order of their administrations. We do not consider medications with precedence constraints in this paper.

III. DISPENSER ARCHITECTURE

Fig. 2 shows the hardware and software components of a smart medication dispenser: They include the medication scheduler, dispenser controller, compliance monitor, network interface and I/O devices, as well as the dispensing unit shown in the upper right-hand corner of the figure.

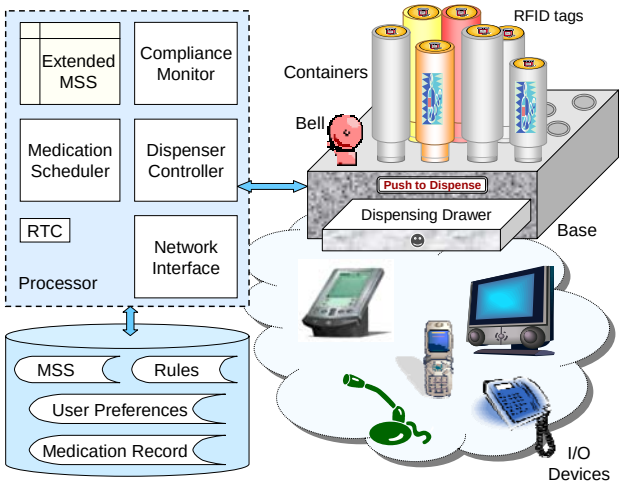

Fig. 2 Architecture and components

A. Dispensing Unit, I/O Devices and Extended MSS

The *dispensing unit* has a base with an array of sockets on the top. Containers, each holding a medication, are plugged in the sockets. Each socket has a *dispensing module* (*DM*). The DM holds the mechanism for releasing the content of the container in the socket. Underneath all the sockets is a *dispensing drawer*; that is where the user retrieves medications. The *Push-To-Dispense* (*PTD*) button in front of the base controls this drawer in the manner described below.

The user is provided with the supply of each medication in a container, with a RFID tag attached. To put new supplies under the control of the dispenser, the user plugs containers into the sockets, one container at a time. The plug-in action causes the dispenser controller to read RFID tags and acquire the name of the medication in the new container upon reading the new tag. After making sure that the MSS includes the direction of the new medication, the controller creates an association between the medication and its socket and maintains the association as long as the container remains

plugged in. As a part of initialization, the controller picks up from MSS information on physical properties of the medication, which the DM of the socket will need.

The number of granules released by DM is determined by the value stored in the *dose-size register* (*DSR*) of the DM. During initialization, the controller clears the DSR of every DM. It loads the dose size into the register immediately before it commands the DM to release a dose. When commanded, the DM releases the specified number of granules and clears the register when it completes.

During normal usage, the dispenser sends a *reminder* to the user shortly before each time for medication. Fig. 2 shows an array of I/O devices. A support infrastructure such as the one described in [24] will allow the dispenser to leverage them in the delivery of reminders to the user in and out of the home. Such connectivity is desirable but not necessary. A minimal dispenser has a local alarm such as the bell on top of the base; an audio interface for delivery of voice reminders, instructions and warnings; a dial-up connection for transmissions of MSS updates and notifications; and a keypad for text input from the user.

The dispenser allows the user to extend the MSS with information on preferences and habits. Armed with this information, the dispenser tries to fit the medication schedule into the daily routine of the user. We assume that the user has configured the dispenser to monitor and record medication history and user behavior. Many behavior parameters affect scheduling. An important one is *promptness*, an estimate of the length of time the user takes to come and retrieve a medication after being reminded that it is time for a dose. Promptness varies, and the user can provide only a rough estimate. By collecting statistics on this parameter, the dispenser will improve the estimate over time.

B. *Dispenser Operations*

The bulk of the work of the dispenser is done by three components: compliance monitor, medication scheduler, and dispenser controller. The compliance monitor maintains medication record and detects and handles non-compliance events. As stated earlier, we do not address this aspect here.

The pseudo code in Fig. 3 gives an overview of operations of the dispenser controller. Initialization is carried out whenever any medication container is added and removed. At the successful conclusion of initialization, the controller calls Schedule.Schedule( ) to start administering the medications.

As its name indicates, the scheduler is responsible for determining the time and dose size of each dose of every medication managed by the dispenser. Schedule( ) and GetNextDose( ) are two of its functions. When invoked, the former computes schedule_table. Similar to schedule tables used in time-driven real-time systems, schedule_table is a list of {time, dose_list} structures, sorted by time in increasing order. time gives the absolute time when some dose of some medication is due, and dose_list provides the names of the medications and their dose sizes to be dispensed at the time. The function GetNextDose( ) returns the {time, dose_list} at the head of schedule_table.

```
Main(…)
{
    NextDoseTime = 0;
    NextDoseList = NULL;
    …
    Initialization( );
    error_occurred = 0;
    error_occurred = Scheduler.Schedule( );
    if (error_occurred != 0) request user attention;
resume:
    dispenser_runs = TRUE;
    while (dispenser_runs == TRUE) {
        error_occurred = Scheduler.GetNextDose(NextDoseTime,
                                                NextDoseList);
        if (error_occurred != 0) break;
        WaitForTimer(NDTT, NextDoseTime – promptness);
        error_occurred = AdministerOperation(NextDoseTime,
                                              NextDoseList);
        if (error_occurred != 0) break;
    }
    if (error_occurred != 0) {
        if (error recovery is successful) goto resume;
    }
    clean up;
    return;
}
AdministerOperation(DoseTime, DoseList) { … }
ReleaseOperation(DoseList) { … }
…
```

Fig.3 Dispenser operation

The dispenser controller works with three timers: Next-Dose-Time-Timer (NDTT) times when the next dose is to be dispensed. While the dispenser runs, the controller repeatedly sets NDTT to expire at a short time before the instant for the next dose of some medication. The length of this short time is equal to promptness. Whenever NDTT expires, the controller carries out the administer operation described in Fig.4. The operation maintains and uses the Wait-for-Retrieval-Timer (WFRT) to limit the length of time the dispenser will wait for the user to respond to a reminder. Release-Mechanism-Timer (RMT) is the third timer. It is used to limit the length of time the dispenser wait for DM's to complete their operations.

ReleaseOperation(DoseList)

1. For every medication in the DoseList, puts the corresponding dose size in the DSR of the DM for the medication.
2. Broadcasts a release command to all DM's.
3. Sets the RMT and waits for timer expiration or all DM's complete.
4. If every DM completes and their DSR cleared before RMT timer expires, clears RMT, records current time, opens the dispensing drawer and plays instructions (e.g., take with a glass of water), if any, and returns 0 to indicate success. If RMT times out, records socket numbers of malfunctioned DM's and error conditions and returns error code.

AdministerOperation(DoseTime, DoseList)

1. Sounds a voice reminder and reminder bell, sets WFRT to expire at DoseTime + MAT, and waits for either WFRT timer expiration or the event that PTD button is pressed.
2. When awaken,
   A. If user has pressed the PTD button, cancels the WFRT timer, and commands a release operation.
      (1) If the release operation succeeded, sends the time recorded by the release operation to compliance monitor along with dose list for logging. Returns 0.
      (2) If release operation failed, picks up error log and returns error code.
   B. If WFRT expired, logs the timer expiration; returns error code.

Fig. 4. Administer and release operations

The administer operation takes as input DoseTime and

DoseList. The former is the time for the current dose, i.e., the dose to be dispensed at the time. In Fig. 4, MAT stands for *maximum allowed tardiness*: If the user does not respond by DoseTime + MAT, the medication scheduler is invoked to re-compute the schedule. We will define the parameter precisely at the end of Section VI.

The administer operation in turn invokes the release operation, which is also described in Fig. 4. We recall the Push-to-Dispense (PTD) button mentioned earlier. The user responds to reminders by pressing the button. If no medication is due at the time when the button is pressed, the dispenser controller keeps the drawer closed and informs the user by voice that no medication is due until some specified time. If some medications are due when the button is pressed, the release operation is invoked. As the result of a successful release operation, the user is given the correct dose of each medication to be taken at the time in the dispensing drawer.

By examining the error code and log, the controller can determine how to recovered from a failed administer operation. A well-built dispenser should rarely, if ever, fail in Step 2A of administer operation, but error due to user's tardiness (i.e., in step 2B) can occur from time to time during normal usage for most users. When this error occurs, the controller calls the scheduler to re-compute the medication schedule and then repeats the administer operation at a later, newly scheduled time. As shown in [7], it is often possible to avoid non-compliance by re-adjusting the schedule. The dispenser sends warnings and notifications only when non-compliance becomes unavoidable.

IV. CONSISTENCY AND FEASIBILITY

When called for the first time after initialization, the Scheduler.Schedule( ) checks the MSS for consistency and feasibility in three steps:

1. For each medication that has interaction pairs, updates its direction parameters if changes specified by change lists in all its interaction pairs are consistent.
2. Ensures that the direction parameters of every medication are consistent and feasibility.
3. Ensure that the separation constraints defined by interaction pairs are consistent and feasibility.

The MSS, and the parameters and constraints given by it, are *consistent* when the medication scheduler can resolve automatically discrepancies, if any, among the parameters. The parameters, and hence the MSS, are *feasible* if all the constraints defined by them can be met simultaneously by some schedule. Scheduler.Schedule() fails and user attention is requested when the medication scheduler finds the MSS inconsistent or infeasible.

In Step 1, dispensers apply only general common sense rules, rather than medication-specific knowledge-based rules. As examples, if change lists of a medication modify some of its absolute limits, the new limits after incorporating the changes should be at least as stringent as the limits set by the medication's own direction parameters unless the MSS explicitly instructs the scheduler to do otherwise. The scheduler declares the MSS inconsistent when it finds the nominal dose size ranges specified by change lists in interaction pairs of a medication do not overlap, because it cannot resolve the difference without external guidance.

A. Necessary Conditions

Hereafter, we assume that the scheduler has completed Step 1 and parts of Steps 2 and 3 successfully. The direction parameters of every medication satisfy the following conditions:

(1) $D_{min} \le d_{min} \le d_{max} \le D_{max}$

(2) $S_{min} \le s_{min} \le s_{max} \le S_{max}$

(3) $D_{max} \le B$, and $S_{max} \le P \le T_{min}$ if S_{max} and P are finite.

Furthermore, for every pair of medications M and N,

(4) $\sigma_{min}(M, N) \le \sigma_{max}(M, N)$.

(5) Precedence relations are consistent [8].

In addition, the direction parameters of every medication with maximum and/or minimum intake constraints must satisfy the following necessary conditions:

(6) $d_{min} \times \text{Floor}[R/s_{max}] \le B$

(7) $\text{Ceil}[L/d_{max}] \le \text{Floor}[P/s_{min}]$

(8) $B/R \ge L/P$

Floor[x] and Ceil[x] denote the floor and ceiling of x, respectively. It is impossible for the scheduler to make the total intake smaller than the value given by the left-hand side of the inequality in (6). The budget B must be at least equal to the lower bound. The left-hand side of inequality in (7) is the minimum number of doses required to have total size L. The number must be no greater than the maximum possible number of doses in an interval of length P. Finally, (8) requires that the dosage rates given by the maximum intake to be no less than the dosage rate given by the minimum intake.

A *consistent medication* is one whose directions parameters satisfy conditions (1) – (8). In general, these conditions are necessary but not sufficient for feasibility: We know that a medication has no *feasible dosage* (i.e., a combination of dose size and separation meeting all constraints) if its direction parameters do not meet some of the conditions, but meeting all the conditions does not mean that there is feasible dosage. The observation below gives some of the special cases:

Observation 1 *A consistent medication has a feasible dosage if either of the following is true. (a) $L = 0$, or $B = \infty$, or both. (b) d_{max} divides B, R divides P, and the nominal separation range includes R (d_{max} / B).*

The medication in case (a) does not have both minimum and maximum intake constraints. That it has a feasible dosage is obvious. In case (b), the scheduler can dispense periodically doses with total size B every R units of time. It follows from (8) that the minimum intake constraint is satisfied as well.

Below is an example. The parameters satisfy necessary conditions (1) – (8) but not conditions in Observation 1.

Example 3: $D_{min} = d_{min} = 5$; $D_{max} = d_{max} = 6$; $S_{min} = s_{min} = 45$; $S_{max} = s_{max} = 46$; $(B, R) = (10, 79)$; $(L, P) = (17, 159)$

To show that the medication has no feasible dosage, we note that there are three doses within a constraint interval of length 159. The dotted and solid step functions in Fig. 5 depict two possible ways for the scheduler to meet the minimum intake constraint using three doses. The time origin is the time of the first of the three doses. The 3-ASAP (As Soon As Possible) function gives the total intake as a function of time when the scheduler uses size d_{max} at 0, followed by two doses of non-increasing sizes with non-decreasing separations. The one labeled 3-*ALAP* (*As Late As Possible*) gives the total intake when the scheduler uses dose size d_{min} at 0, followed by doses of non-decreasing sizes with non-increasing separations. The intake function of any schedule that dispenses three doses with total size 17 lies in the shaded region bounded by these two functions. We can think of intake as the required dosage demand. The maximum intake (10, 79) constraints the supply; it is depicted by the heavy dashed lines. The fact that the dashed line lies below the intake functions sometimes indicates that the supply is insufficient to meet the demand at all times and, therefore, the medication has no feasible dosage.

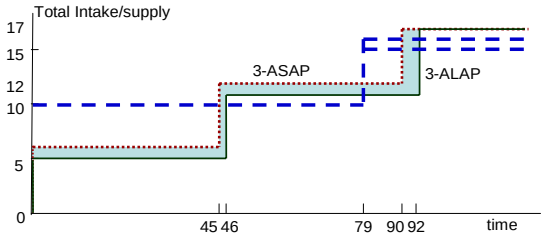

Fig. 5 A Feasibility test

B. *Dosage Demand Analysis*

Fig.6 describes a feasibility test motivated by this example. It is called *Dosage-Demand Analysis* (*DDA*). The input for DDA consists of nominal dosage and intake parameters (i.e., d_{min}, d_{max}, s_{min}, s_{max}, B, R, L, and P) of the medication under test. If the parameters satisfy the necessary conditions listed above but not the conditions in Observation 1, the test starts from the minimum number of doses required to get a total dose size L and tries every possible number of doses. As soon as it finds an ALAP intake function that never exceeds the dosage supply in the interval $[0, P)$, it concludes that the medication may be feasible. It declares the medication infeasible if it finds no such intake function after trying all possible numbers of doses.

DDA uses the function ALAP_Doses_Separations( ), which takes as an input the number K of doses to be dispensed in a minimum intake constraint interval. If successful, the function returns the arrays dose_size[K] and separation[K] based on the ALAP strategy. Part (b) of Fig. 6 describes the function.

Let $t_0 = 0$, and t_j, for $j = 1, \ldots K\text{-}1$, be the time of the j-th dose. By definition, t_j is equal to the sum of separation[k] for $1 \le k \le j$. The value of *intake*(*0*) is dose_size[0], and *intake*(*t*) increases by dose_size[j] at t_j. The value of the supply function *supply*(*t*) is equal to B at 0. It increases at replenishment time $t = R$ by dose_size[0] and again by dose_size[j] at $t = t_j + R$, for each $j = 1, \ldots K\text{-}1$.

```
Input: Parameters d_min, d_max, s_min, s_max, B, R, L, P
Output: is_consistent = TRUE; is_infeasible = FALSE;
If (any of the conditions (1) – (8) is not satisfied)
    is_consistent = FALSE; return;
if ((M satisfies any of the conditions stated in Observation 1)  return;
max_number_doses  = Ceil [P / s_min];
min_number_doses = Ceil [L / d_max];
number_doses = min_number_doses;
times_doses_chosen = FALSE;
while (number_doses < max_number_doses) {
    times_doses_chosen = ALAP_Doses_Separations(number_doses);
    if (times_doses_chosen == FALSE) {
        break;
    } else {
        compute ALAP intake function Intake(t) for t in [0, P);
        compute dosage supply function Supply(t) for t in [0, P);
        if Intake(t) ≤ Supply(t) for all t in [0, P)  return;
    }
    increment  number_doses by 1; times_doses_chosen = FALSE;
}
Is_infeasible = TRUE;
return;
```

(a)

```
Input:  Parameters d_min, d_max, s_min, s_max, B, R, L, P; Number of doses K;
Output: dose_size[K] = {d_min}; separation[K] = {s_min}; succeeded = TRUE;
remaining_dose = L ; remaining_interval = P – 1;
for (index = 1; index <= K; index = index + 1) {        // Select dose sizes.
    dose_size[K – index] = remaining_dose – (K – index) * d_min ;
    if (dose_size[K – index] < d_min) return succeeded = FALSE;
    if (dose_size[K – index] = d_min) {
        break;
    } elseif (dose_size[K – index] > d_max) {
        dose_size[K – index] = d_max;
    }
    remaining_dose = L – dose_size[K – index] ;
}
separation[0] = 0;                                      // Select separations
for (index = 1; index < K; index = index + 1) {
    separation[index] = remaining_interval – (K – 1 – index) * s_min ;
    if (separation[index] < s_min) return succeeded = FALSE;
    if (separation[index] = s_min ) {
        break;
    } elseif (separation[index] > s_max) {
        separation[index] = s_max;
    }
    remaining_interval = P – separation[index];
}
return succeeded;
```

(b)

Fig. 6 Dosage Demand Analysis (DDA)

To illustrate DDA, suppose that the ranges of dose size and separation in Example 3 are widen so that $d_{min} = 3$, $d_{max} = 10$, $s_{min} = 45$, and $s_{max} = 80$. It is possible to use two doses in a minimum intake constraint interval of length 159. Initially, the dose-separation selection function sets both elements of dose_size[ ] to the minimum dose size 3. In the first iteration of select-dose-size loop, dose_size[1] is set to 10, which is the minimum of $d_{max} = 10$ and $L - d_{min} = 14$. dose_size[0] is then set to 7. Select-separation loop computes only separation[1] and sets it to $min(80, 158 – 45) = 80$. In this case, the 2-ALAP intake function never lies above the supply function. This fact suggests that some dosage may be able to meet both the minimum intake and maximum intake constraints. Indeed a dose of size 10 every 79 units of time is such a dosage.

The DDA test resembles the time-demand analysis method [19] used for determining the schedulability of fixed priority periodic tasks scheduled on a processor. The scheduler only needs to check whether the supply meets the demand at time

instants where the intake function or supply function has step increases. The number of times for this check is 2 K. There is an important difference however. In time-demand analysis, time supply increases with time independent how the processor is allocated to tasks. In contrast, dosage supply at time t depends on the sizes and times of the doses dispensed before t. In this respect, DDA resembles slack time estimation in fixed priority systems (e.g., [21]).

V. Dosage Selection

Before computing a medication schedule, the scheduler selects for each medication, from the respective nominal ranges given by the MSS, a feasible dosage (i.e., values of dose size d and separation s) it will use in scheduling the medication. An algorithm used for this purpose is called a *dosage selection algorithm*.

A. Heuristic Algorithms

The previous section says that dosage selection can fail in the general when the medication has both intake constraints. Table 1 lists two families of heuristic dosage selection algorithms for the general case: *Independent algorithms* make independent selections of the sizes and separations for individual doses. The four in the first column select boundary values: Their selections of dosage (d, s) are (d_{max}, s_{min}), (d_{min}, s_{max}), (d_{max}, s_{max}) and (d_{min}, s_{min}), respectively, for all doses.

TABLE 1 Dosage selection algorithms

Independent		Intake-Guided	
Maximum	Average	MaxD_AMAP	MaxD_ALAP
Minimum	Uniform	MinD_AMAP	MinD_ALAP
Likely_Large	Random_Large	AveS_AMAP	AveS_ALAP
Likely_Small	Random_Small	AveD_AMAP	AveD_ALAP

The Average Algorithm uses average dose size and average separation. The last three in this group make independent random selections for each dose. Uniform Algorithm selects for d and s from uniform distributions over the respective ranges of the parameters. Random_Large and Random_Small randomly select d and s from right triangle shape probability density functions (pdf) over the dose-size and separation ranges. For the former, the right angles of the dose-size and separation pdf's are at d_{max} and s_{min}, respectively, while for the latter, they are at d_{min} and s_{max}.

In contrast, *intake-guided algorithms* make correlated selections. Each of these algorithms starts by fixing either dose size or separation and then uses one of the intake constraints to guide the selection of the second dosage parameter. The first parts in the names of these algorithms tells us their choices of the first parameter: MaxD, MinD, AveD and AveS indicate that the choices of the algorithms are $d = d_{max}$, $d = d_{min}$, $d = (d_{max} + d_{min})/2$, and $s = (s_{max} + s_{min})/2$, respectively.

The algorithms with AMAP (As Much As Possible) in their names use the maximum intake (B, R) to guide the selection of the second dosage parameter. In essence, the algorithms try to make the total dose size in each interval of length R as close to the upper limit B as possible, hence the name. When the dose size d is selected first, the algorithms selects as separation s the value in the nominal separation range closest to Ceil[R / Floor[B/d]]. If s is first selected, d is the nominal dose size closest to Floor[B / Ceil[R/s]]. Algorithms with ALAP (As Little As Possible) in their names use the minimum intake (L, P) as the guide; they try to make the total size of doses in each interval P as close to the lower limit L as possible. If s is first selected, then d is the nominal dose size closest to Ceil[L / *Floor*[P/s]]. If d is first selected, s is the nominal separation closest to Floor[P / Ceil[L/d]].

B. Performance

The determine their performance, we evaluated the algorithms in Table 1 by using them to select dosages for "medications" with randomly generated direction parameters. The heuristics are compared according to two criteria. First, *success rate* is the fraction of all medications for which the algorithm succeeded in finding feasible dosages. A selected dosage gives the scheduler more leeway in scheduling if it allows larger deviation from the selected separation s: The larger the allowed deviation, the better. We use *usable separation range to* quantify this aspect: It is the width of the allowed deviation, normalized with respect to the width of the given nominal separation range.

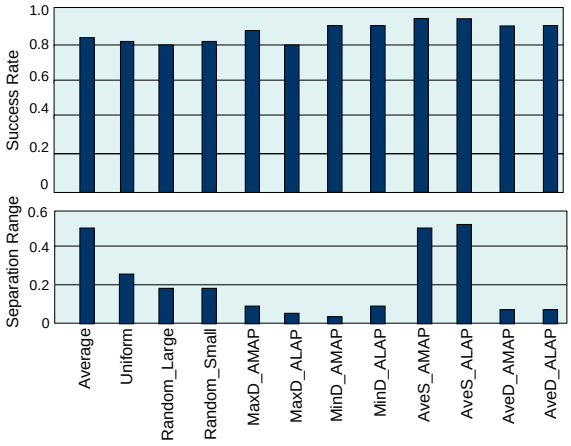

Fig. 7 Relative performance of dosage selection algorithms

The performance summary in Fig.7 was obtained from sample medications whose parameters were generated in the following manner: For each sample, d_{max} and s_{max} were chosen first independently from even distributions [1, 30] and [1, 1440], respectively. We use two ratio parameters r_d and r_s to determine the lower boundaries of dose size and separation ranges: $d_{min} = d_{max}(2\text{-}r_d)/(2+r_d)$ and $s_{min} = s_{max}(2\text{-}r_s)/(2+r_s)$. r_d and r_s were initialized as 0 and incremented independently by 0.1 per step until 2.0 and 1.9, respectively. Intake parameters B and L are random multiples of d_{max} and d_{min}, respectively. Their respective multipliers r_B and r_L are independently selected from the even distribution over the range [1, 100]. Similarly, P and R are random multiples of s_{max} and s_{min}, respectively. Their respective multipliers r_P and r_R are also

independently selected from the even distribution over the range [1, 100]. We threw away samples that were found inconsistent, and the number of remaining samples was large enough to yield 10% statistical error.

From Fig. 7, we can see that the success rates of almost all algorithms are acceptably good. They differ noticeably in usable separation range, however. (The reason for leaving out the algorithms listed in the first column of Table 1, despite their good success rates, is that their usable separation ranges are equal to 0.) In general, a consequence of intake constraints is a significant reduction in separation range.

VI. Medication Scheduling

When the scheduler starts to compute a medication schedule, it has already selected a feasible dosage for every medication M. Since there is no possibility of confusion, we denote the usage separation range also by $[s_{min}(M), s_{max}(M)]$) and call them nominal separations. The dosage selection process has also made sure that that the scheduler can always find a size for each dose. Hereafter, we focus solely on the problem of choosing times of the doses, i.e., scheduling.

A. Resource and Workload Models

The medication scheduler is concerned solely with scheduling the only scarce resource, the user. It uses two types of virtual resources to manage contention for the user [7] and allocates these entities as if they were physical processors and resources: There is a processor P_M for every medication M. The scheduler uses P_M to maintain correct separation between doses of M. If M interacts with some other medication or with food, then there is also a resource R_M for M. The scheduler uses the resource R_M in its effort to maintain the required minimum separation between doses of M and doses of other medications and food.

Processor Scheduling It is convenient to view doses of M as non-preemptable jobs $J_M(i)$, for $i = 1, 2, \ldots$, on P_M. A job *starts* at the time when the dose it models is released. The job occupies the processor P_M (i.e., the user) for $s_{min}(M)$ units of time. In terms from real-time systems literature, $e_M = s_{min}(M)$ is the *execution time* of (jobs of) M.

The first job of each medication M is scheduled to start at some specified time or on best effort basis. The start times of subsequent jobs of M are constrained by three types of deadlines: First, the *inter-stream relative deadline* of $J_M(i)$ with respect to the previous job $J_M(i\text{-}1)$ of M is equal to their absolute maximum separation $S_{max}(M)$. Second, $J_M(i)$ may be required to start by the end of the current minimum intake interval. This requirement imposes on the job an *effective minimum intake deadline*. Third, if doses of M must be taken sufficiently close together with doses of other medications, the inter-medication maximum separations imposes on $J_M(i)$ one or more *inter-stream separation deadlines*. The absolute start time deadline of $J_M(i)$ is the earliest absolute deadline computed from these relative deadlines. Precise definitions and illustrative examples can be found in [7].

A job is said to be *precisely scheduled* when the time allocated to it by the scheduler is equal to its execution time. The medication scheduler always starts with a *feasible precise schedule* according to which all jobs start within their deadlines and are precisely scheduled.

A typical user is not always prompt. A larger than expected promptness may cause some jobs to start later than their scheduled times. As a consequence, there may not be enough time to give some job $J_M(i)$ $s_{min}(M)$ units of time before the subsequent job of M must start. When this occurs, the scheduler treats the job as an *imprecise job* [14], consisting of a mandatory part followed by an optional part. The execution time of the mandatory part is equal to the absolute minimum separation $S_{min}(M)$. The scheduler may allocate the optional part less than $e_M - S_{min}(M)$ units of time when user tardiness forces it to short change the part in order to enable the on-time start of the subsequent jobs of M.

The observations below follow from the definitions of jobs, their processor time requirements, start time deadlines and scheduling rules [7].

Observation 2 *All separation constraints between doses of M are met when every job of M starts within its deadline $S_{max}(M)$ relative to the start time of the previous job of M and its mandatory part is precisely scheduled on P_M.*

Observation 3 *Absolute maximum separation constraints specified by all the interaction pairs are met when every job starts by its absolute deadline and its mandatory part is precisely scheduled on P_M.*

Resource Allocation The scheduler follows two rules in allocation of R_M: (1) Each job of M must have R_M exclusively for an infinitesimally small amount of time in order to start. (2) Every job of N ($\neq M$) in each interaction pair $I(M, N)$ of M requires the resource R_M on a shared basis for $\sigma_{min}(N, M)$ units of time beginning from when the job starts.

Because of Rule (1), the resource R_M serves as a permit for jobs of M. A job of N can block a job of M for $\sigma_{min}(N, M)$ units of time. We call this time *blocking time* of M by N. The *worst case blocking time* of a medication M is equal to the maximum over all $N \neq M$ the blocking time of M due to N. In contrast, an arbitrary number of jobs of medications other than M can share the resource R_M. The observation below follows as a direct consequence.

Observation 4 *Minimum separation constraints specified by all interaction pairs are met when resources are allocated to jobs according to Rules (1) and (2).*

B. Scheduling Algorithms

We have been experimenting with scheduling algorithms based on the above described model. All the algorithms assign priorities to jobs. There are two variants: non-greedy and priority-driven. A *non-greedy algorithm* may choose to let jobs wait intentionally. A *priority-driven algorithm* never let any processor or resource idle intentionally.

MVF Scheme Among the priority schemes we have studied, the MVF (Most-Victimized-First) schemes seems to be a natural choice for scheduling interacting medications. MVF priority scheme gives fixed priorities to jobs based on their worst case blocking times; the longer the worst case blocking time, the higher the priority: Jobs of the same medications are scheduled in FIFO order. Using the non-greedy version, the scheduler considers one medication at a time in priority order and generates a complete schedule of the medication. According to the priority-driven variant, ready jobs are scheduled one at a time without look ahead. The scheduler views the sequence of jobs of each medication as a periodic task [8]. The length of intervals between consecutive jobs of the task is in the range $[s_{min}(M), s_{max}(M)]$.

To illustrate, we consider a user who takes the medications named in the graph in Fig. 8. The graph, called a separation graph [7], has a node for each medication and food. The square bracket under the medication name gives its usable separation range. There is an edge from M to N with label $\sigma_{min}(M, N)$ if $\sigma_{min}(M, N) \neq 0$. (Maximum separations between medications are all zero in this example.) The worst case blocking times of Fosamax, antibiotic, food and vitamin are 6, 2, 1.0 and 0.5, respectively. Hence, the MVF scheme gives jobs of Fosamax the highest priority, followed by jobs of antibiotic, meals and then by jobs of vitamin.

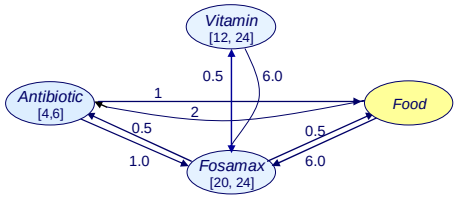

Fig. 8 A separation graph

Fig. 9 shows segments of schedules produced by three variants of MVF scheme: non-greedy MVF (MVF-NG); MVF-NG-Food-First (MVF-NG-FF); and the priority-driven version (MVG-PD). All three schedules are periodic with a 24-hour period. The time origin is the start of a day. In the figure, P_{FX}, P_A, P_{FD}, *and* P_V refer to processors for Fosamax, antibiotic, food and vitamin, respectively; their resources are named R_{FX}, R_A, R_{FD}, *and* R_V, respectively. The boxes on a timeline labeled by a processor name indicate the time intervals during which the processor is in use.

Following the MVF-NG algorithm, the scheduler treats food as if it were a medication. It begins by scheduling jobs of Fosamax. It then schedules jobs of antibiotic, meals and vitamin in order. The Fosamax job (i.e., dose) of day starts at 0 and is allocated R_A, R_{FD}, and R_V, each for 0.5 hour. The job occupies the processor P_{FX} for 20 hours. The first antibiotic job cannot start until 0.5, since R_A is not available until then. The scheduler decides to use the maximum nominal separation and schedules the 4 jobs of the day evenly space in time at 0.5, 6.5, 12.5 and 18.5. Each job occupies P_A for 4 hours and is allocated R_{FX} and R_{FD} for one hour. The user cannot eat the first meal until time 1.5 when R_{FD} becomes available. Each food job occupies P_F for 0.5 hr. Given the schedule of subsequent antibiotic jobs, the scheduler schedules the second and third meals at 7.5 and 13.5. Finally, since only Fosamax blocks the vitamin job of the day, the job can start at time 0.5.

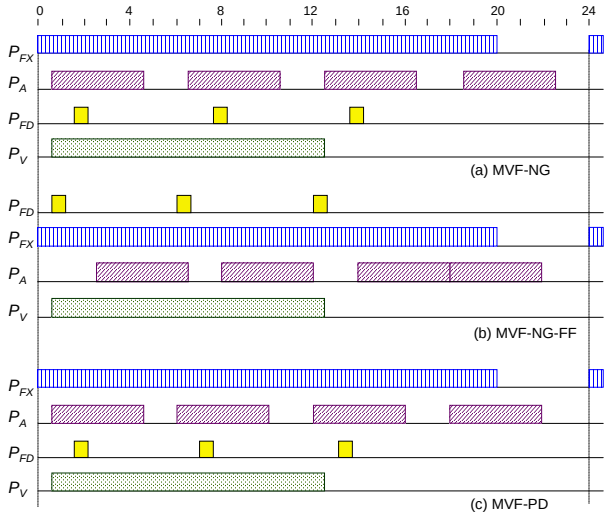

Fig. 9 MVF schedules of interacting medication

While MVF-NG algorithm treats food a medication, MVF-NG-FF algorithm plan meals times based on user preference and then schedules jobs of medications in time intervals allowed. This is the strategy advocated in [6]. In essence, meal times specified by the user are treated as forbidden intervals for medications that interact with food. It alters the forbidden intervals only when changes are necessary. Suppose that the start of the day is 6 AM. The user prefers to have breakfast at 6:30 AM, lunch at noon and dinner at 6PM, as well as uninterrupted sleep for 6 hrs. The MVF-NG-FF schedule in Fig. 9 is generated to fit these user preferences.

The MVF-PD schedule shown in Fig. 9 is generated using the priority-driven variant. The job in each period is ready for scheduling at the beginning of the period. In this example, periods of the Fosamax, antibiotic, and vitamin are 24, 6, and 24, respectively. At time 0 (the start of the day), the tasks are in-phase (i.e., a period of every task starts at 0.). Suppose that according to the user preference, the food jobs of the day become ready at 0 or 0.5, 6 and 12. We note that the MVF-PD schedule is similar to the MVF-NG schedule except for the large jitter of the antibiotic job. It is acceptable only when the absolute maximum separation is 6.5 or more. Compared with RM-PD and EDF-PD schemes however, MVF-PD is much superior; the rate-monotonic and EDF priority schemes fail to produce acceptable schedules in this case [7].

Performance Measures We measure the merit of a medication scheduling algorithm by the quality of the schedules it produces. Quality of medication schedules has two dimensions: adherence to medication directions and user friendliness. Several figures of merit quantify how close a schedule adheres to medication directions. They include dose-size variation, separation jitter, and deviation from nominal parameter ranges [7]. By these criteria, the MVF-NG

schedule in Fig.10 is almost ideal: If the user is prompt, there is no separation jitter and no deviation from nominal separation for every medication. However, keeping a constant separation between doses is also why the schedule is not user friendly, with poor meal times and sleep times. In contrast, separation jitters of antibiotic are 2.25 and 0.5, respectively, for the MVF-NG-FF and the MVF-PD schedule. By this criterion, MVF-NG-FF performs poorly, especically if the user's condition treated by the antibiotic is serious enough to warrant a relative constant level of the medication.

We use the maximum allowed tardiness as a measure to quantify user friendliness. *Tardiness* is the difference between the actual promptness and the estimated promptness, if the difference is positive and is equal to 0 otherwise. The *maximum allowed tardiness* (*MAT*) of a dose *i* of medication *M* according to a schedule is the maximum length of time the start time of the corresponding job $J_M(i)$ can be delayed without violation any hard constraints. The *MAT* of the medication *M* is the minimum *MAT* over all doses of *M*. Take the MVF-NG-FF schedule in Fig. 9. Suppose that the absolute maximum separation of antibiotic is 9 and the minimum separation of antibiotic is 3. Then the values of *MAT* of the 4 daily doses of antibiotic are 0.5, 3, 1 and 5. Therefore, the *MAT* of antibiotic is 0.5. According to the MVF-NG schedule, the *MAT* of antibiotic is 0. (A delay in any of the first three doses of antibiotic would violate the minimum separation constraint between the medication and food.) By this criterion also, MVF-NG is not user friendly.

VII. Summary

We describe in this paper architecture of smart medication dispensers and algorithms for consistency checking, dosage selection and scheduling. In addition to building a prototype dispenser, we are evaluating these and other medication scheduling algorithms, to determine their ability to produce feasible schedules and other quality measures, on real-life and synthetic medication schedule specification. Much work in this direction remains to be done. Compared with what we know about algorithms for scheduling real-time tasks, we are still far from having the necessary insight and understanding about the behavior of medication scheduling algorithms. We also need to have sound graceful degradation mechanisms. We discussed earlier the application of imprecise computation. We will explore in depth its use, as well as the use of other schemes (e.g., [25]) for prevention of non-compliance.

Acknowledgement

This work is partially supported by the Taiwan Academia Sinica Thematic Project SISARL: Sensor Information Systems for Active Retirees and Assisted Living. The authors wish to thank Dr. D. Burkhardt and Mr. T. S. Chou for their suggestions on medication specification and dispenser design.

References

[1] S. C. Dursco, "Technological Advances in Improving Medication Adherence in the Elderly," *Annals of Long-Term Care: Clinical Care and Aging*, Vol. 9, No. 4, 2001.

[2] http://www.dynamic-living.com/automated_medication_dispenser.htm and http://www.epill.com/

[3] My Pill Box at http://www.mypillbox.org/mypillbox.php

[4] Harris Interative, "Prescription Drug Compliance a Significant Challenge for Many Patients," http://www.harrisinteractive.com/news/allnewsbydate.asp?NewsID=904, March 2005.

[5] A. V. Law, M. D. Ray, K. K. Knapp, and J. K. Balesh, "Unmet Needs in Medication Use Process: Perceptions of Physicians, Pharmacists, and Patients," *Journal of American Pharmaceutical Association,* 2003.

[6] P. C. Hsiu, H. C. Yeh, P. H. Tsai, C. S. Shih, D. H. Burkhardt, T. W. Kuo, J. W. S. Liu, T. Y. Huang, "A General Model for Medication Scheduling," Institute of Information Science, Academia Sinica, Taiwan, Technical Report TR-IIS-05-008, July 2005.

[7] P. H. Tsai, H. C. Yeh, P. C. Hsiu, C. S. Shih, T. W. Kuo, J. W. S. Liu, "A Scarce Resource Model for Medication Scheduling," Institute of Information Science, Academia Sinica, Taiwan, Technical Report TR-IIS-06-003, April 2006.

[8] J. W. S. Liu, *Real-Time Systems*, Chapters 2 – 4, Prentice Hall, 2000.

[9] B. Spruri, L. Sha, J. P. Lehoczky, "Aperiodic task Scheduling for Hard Real-Time Systems," *Real-Time Systems Journal*, Vol. 1, No. 1, 1989.

[10] T. M. Ghazalie and T. P. Baker, "Aperiodic Servers in Deadline Scheduling Environment," *Real-Time Systems Journal*, 1995.

[11] T. S. Tia, J. W. S. Liu and M. Shankar, "Algorithms and Optimality of Scheduling Aperiodic Requests in Fixed Priority Preemptive Systems," *Real-Time Systems Journal*, Vol. 10, No. 1, January 1996.

[12] J. P. Lehoczky, L. Sha, and Y. Ding, "The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Behavior," *Proceedings of IEEE Real-Time Systems Symposium*, December 1989.

[13] N. Audsley, A. Burns, K. Tindell, M. Richardson, and A. Wellings, "Applying a New Scheduling Theory to Static Priority Preemptive Scheduling," *Software Engineering Journal,* Vol. 5, No. 5, 1993.

[14] J. W. S. Liu, K. J. Lin, W. K. Shih, R. Bettati and J.Y. Chung, "Imprecise Computations," *IEEE Proceedings*, January 1994.

[15] C.C. Han, K.J. Lin and J.W.S. Liu, "Scheduling jobs with temporal distance constraints," *SIAM Journals on Computing,* 1995.

[16] G. Gerguson, J. Allen, N. Blaylock, D. Byron, N. Chambers, M. Dzikovska, L. Galescu, X. Shen, R. Swier, and M. Shift, "The Medication Advisor Project: Preliminary Report," Report No. 776, Computer Science Department, University of Rochester, May 2002.

[17] D. M. Cutler, N. E. Feldman, and J. R. Horwitz, "U. S. Adoption of Computerized Physician Order Entry Systems," *Health Affairs*, 2005.

[18] R. W. Park, S. S. Shin, Y. I. Choi, J. O. Ahu, and S. C. Hwang, "CPOE and Electronic Medical Record Systems in Korea Teaching Hospitals," *Journal of American Medical Informatic Association,* 2005.

[19] R. L. Davis, "Computerized Physician Order Entry Systems: The Coming of Age for Outpatient Medicine," PLoS Medicine, 2005.

[20] B. Koppel, *et al*., "Role of Computerized Physician Order Entry Systems in Facilitating Medication Errors," *Journal of AMA*, 2005.

[21] Health Information Systems: http://www.hhs.gov/healthit/ahic.html

[22] PDRHealth, Drug Information, http://www.pdrhealth.com/drug_info/

[23] 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," to appear in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics,* October 2006.

[24] Q. Wang, W. Shin, X. Liu, Z. Zeng, C. Oh, B. K. Alshebli, M. Caccamo, C. Gunter, E. Gunter, and J. Hou, "An Open Architecture for Assisted Living," to appear in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics,* October 2006.

[25] M. Hamdaoui and P. Ramanathan, "A Dynamic Priority Assignment Technique for Streams with (m, k)-firm deadlines," *IEEE Transactions on Computers*, Vol. 44, No. 12, December 1995.