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

TR-IIS-10-008

A Model and Simulation Environment for Symbiotic Automation and Assistive Devices

P. H. Tsai, C. H. Chen, C. W. Yu, C.S. Shih and J. W. S. Liu

Oct. 26, 2010 || Technical Report No. TR-IIS-10-008

http://www.iis.sinica.edu.tw/page/library/TechReport/tr2010/tr10.html

Institute of Information Science, Academia Sinica
Technical Report TR-IIS-10-008

A Model and Simulation Environment for Symbiotic Automation and Assistive Devices

T. Y. Chen, P. H. Tsai, C. H. Chen, C. W. Yu, C.S. Shih and J. W. S. Liu

Abstract

This paper describes the UCAADS simulation environment and the underlying UCAADS model that have been developed for the purpose of evaluating the correctness and performance of UCAADS and user-device interactions. The acronym UCAADS stands for *user-centric automation and assistive devices/systems and services*. Examples of UCAADS are medication dispenser, smart pantry, robotic housekeeping aids, and mobility assistants. UCAADS model combines two types of modeling elements: workflow model and GOMS model. The behavior specification of a symbiotic system of device and its user(s) as a whole consists of specifications of device operations, user actions and user-device interactions. They are defined in terms of workflows and are executable. The incorporation of GOMS model with workflows enables us to account for different behavior and skill levels of different users in the estimation of execution times of their actions. As case studies, we modeled and simulated parts of three UCAADS: smart medication dispenser for home use, smart storage pantry and multi-user medication station. These devices require their users to carry out mission-critical operations. Our simulation experimentations and the results demonstrate that the UCAADS model and USE are effective in helping us discover and fix design and implementation errors that allow incorrect user-device interactions, in addition to assessing the responsiveness of devices.

T. Y. Chen, C. H. Chen and C. W. Yu, Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. Their email addresses are yen@iis.sinica.edu.tw, rustic308@gmail.com and cwyu.cs@gmail.com; C. S. Shih, Department of Computer Science and Information Engineering, National Taiwan University, Taiwan. Email: cshih@csie.ntu.edu.tw; P. H. Tsai and J. W. S. Liu are affiliated with Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan. Email: peipei@iis.sinica.edu.tw, janeliu@iis.sinica.edu.tw

Table of Contents

1 Introduction

In the coming decades, we are likely to witness accelerated growth in diversity and use of *user-centric automation and assistive devices/systems and services (UCAADS)*. Some UCAADS are home automation and assistive devices and services targeted for users who are elderly or have functional limitations. Such devices intend to improve the quality of life of their users, enable the users to live independently longer, make the user's physical therapy regiment more effective, and so on. Examples are smart storage pantry, object locator, and smart medication dispenser [1-5], and robotic housekeeping aids and mobility assistants [6-14]. Other UCAADS include automation tools and equipment for care-providing institutions. Examples are systems and devices that assist care providers in medication dispensing and administration and automate these stages of the medication use process to the desired extent (e.g. [15, 16]).

Despite the vast differences in the functionalities and appearances of UCAADS, these devices share many commonalities. First and foremost is that they are user-centric. According to the classification proposed in [17], user-centric devices/systems are for discretionary use, versus mandatory use of machine-centric devices (such as autopilot and precision machinery). User-centric devices/systems must be easy to use, configure, customize and maintain. A user-centric device should be *safe*, meaning that the device never does any harm even when misused and erroneous operations are recoverable. Making user-centric devices safe as well as *flexible* (i.e., configurable and customizable) is particularly challenging because the majority of devices/systems exemplified by the ones listed above are semi-automatic. Some of them require their users to carry out critical operations. Being for discretionary use, it is impractical to require more than minimal user training, if any training at all. At the same time, the skills of users may vary widely across user population, and the skill of each individual user may change over time.

This paper presents the UCAADS model and UCAADS simulation environment (USE). USE is designed to help developers assess the usability and flexibility of new or modified devices and

the degree to which the devices are safe. In particular, USE aims to help developers identify and fix design and implementation errors that can cause the device to malfunction or allow unsafe user-device interactions with harmful consequences.

The inputs to USE for the purpose of simulating or emulating a device and its interactions with the user(s) are based on the UCAADS model. The model incorporates two types of elements: workflows [18] and GOMS models [19]. It captures the behavior of a device being evaluated in terms of one or more workflows, called *device workflows*. User actions (operations) are captured by *user workflows*. We refer to the device and user workflows of a device collectively as an *operational specification* of the device. As it will become evident in later sections, operational specifications defined by workflows are expressive and easy to understand. The UCAADS model also incorporates elements of GOMS and MHP (model human processor) [20] as building blocks of user workflows and thus enables the developers to model user actions more precisely. By using workflows to define human operators (i.e., perceptual, cognitive, and motor operators), they also can be executed during simulation for the purpose of estimating execution times of user actions for different users.

In USE, a workflow can be defined in terms of either the languages that are supported by Microsoft workflow foundation [21] (e.g., C# and BPEL [22]) or SISARL-XPDL [23] that extends the WfMC (Workflow Management Coalition) [24] standard XPDL (XML Process Definition Language) [25] with elements needed for embedded and robotic applications. USE uses workflow engines [21, 23] (i.e., a middleware) for execution of workflows. For the case studies described in this paper, USE ran on top of Microsoft workflow foundation (WF), and operation specifications were executed by the WF workflow engine. When the workflow-based implementation of UCAADS device in term of workflows written in SISARL-XPDL becomes available, USE provides a compiler for translate workflow definitions into executable scripts and executes the scripts on EMWF (Embedded Workflow Framework) [23].

In addition to supporting the execution of the device workflows and user workflows in the operational specification of a device being evaluated, USE also provides the developer with an easy-to-use interface to specify events to be captured, analyzed, displayed and recorded during simulation. After studying the recorded data, the developer may want to refine the design and operational specification of the device. USE provides extendible libraries of reusable model components to reduce the cost and effort in model construction and refinement.

The remainder of the paper is organized as follows. Section 2 presents closely related works. Section 3 provides brief descriptions of three UCAADS. They are subjects of case studies presented in Section 6. Section 4 presents workflow and GOMS elements of the UCAADS model. Section 5 describes the architecture and components of USE and how the UCAADS model and USE are used in the development process of UCAADS. Specifically, we use the above mentioned devices as case studies and present simulation results on their performance in Section 6. Section 7 summaries the paper and presents future extensions of the UCAADS model and USE.

2 Related Works

Again, the UCAADS model and USE build on advances in workflow technology [18] and GOMS models [19]. USE resembles many simulators, prototyping tools and development environments in their objectives. We all aim to facilitate the evaluation of device designs and implementation throughout the development process.

2.1 Workflow Technology

The workflow technology is commonly used in enterprise systems for automation of business processes. Recently, light weight workflow management systems [26-28] enables the workflow technology be applied to build mobile web-based applications, embedded devices and robotic applications [23].

There are many reasons for the wide adoption of this technology. First and foremost,

workflow provides an easy, flexible way to define complex business processes. The developer can design and implement a new workflow application or reconfigure and customize an existing one by supplying or modifying definitions of workflows in it. Existing standard workflow process definition and execution languages (e.g., [22, 25, 29, 30]), together with tools (e.g., [21, 31-35]) for editing workflow definitions and for parsing and building them, significantly reduce the effort to do these tasks. USE is particular suitable for evaluation of workflow-based devices since their operational specifications generated at the design stage become implementation of the device later in the development process.

2.2 GOMS Models and Tools

The well-known human processor model GOMS (Goals, Operators, Methods and Selection Rules) has been widely used for years in studies on human-computer interactions [36, 37]. According to GOMS, each user action, usually referred to as a task, has one or more goals. The action is composed of operators that are done according to specified methods to achieve the goal(s). When there is more than one method applicable to a goal, the selection rules represent the user's knowledge of which method should be applied.

Four variants of GOMS (i.e., CMN-GOMS, KLM, NGOMSL and CPM-GOMS) [20, 38-40] have been used for predictions of user performance and evaluation of user interface [41-43]. USE uses NGOMSL and CPM-GOMS. Their usages will be illustrated by case studies presented in Section 6. According to the former, human operators are executed sequentially. The latter makes the assumption that perceptual, cognitive and motor operators can be performed in parallel. We use NGOMSL because it is easy to translate NGOMSL analysis into workflow model and use CPM-GOMS because it allows us to model complex user actions.

It is time-intensive and labor-intensive to construct GOMS models by hand. Several software-tools have been developed to ease the burden of building GOMS models. QGOMS, CAT-HCI, GLEAN, CRITIQUE [44-47] are among the pioneers of tools. Other examples are

Apex-CPM [48] and SANLab-CM [49]. USE most closely resembles CogTool [50], which is a UI prototyping tool. Both are developed for rapidly prototyping, evaluation of design and prediction of human performance. Using CogTool to evaluate a UI design, the designer presents the design as a storyboard of frames. Each frame represents a state of the interface, and each transition between frames represents user actions that take the UI from one state to another. To analyze the design, the designer demonstrates tasks on the storyboard. CogTool automatically generates ACT-R code [51] implemented KLM models from this demonstration. By running the code, CogTool produces an estimate of execution time and a visualization of the timeline.

A major difference between USE and all the tools mentioned above is that unlike them, USE is not primary for the evaluation of user interfaces. Rather, it supports the evaluation of user-device interactions in general for the purpose of determining the correctness and usability of a device/system throughout the development process, from prototyping to implementation to quality assurance. Similar to above mentioned HCI tools, USE also provides a rich and extensible library called *Human Action Library*. The library includes operators and templates of GOMS in the form of activities and workflows to significantly reduce the burden of building GOMS models for execution in USE. USE can be easily hook up with Window Form and WPF (Windows Presentation Foundation) [52] and is hence particularly convenient for evaluating user interfaces implemented using them.

2.3 Prototyping and Simulation Tools

Simulation is an effective method for many purposes. Over time, simulation has become widely used for reducing costs and improving qualities for an increasing broader spectrum of devices, system and services. Today, one can find simulation environments and simulators for networks (e.g., [53]), sensor networks (e.g., [54]), embedded systems (e.g., [55]), and robotic applications (e.g., [56, 57]).

Among prototyping and simulation tools, USE resembles closely to toolkits such as D.tools

[58] and Juxtapose [59] that support iterative-design-centered approach to prototype physical user interfaces, in particular, design exploration of desktop, mobile and physical interfaces of interactive devices. The USE has the same design goals but focuses especially on user-device interaction and flexible design and implementation of UCAADS. USE distinguishes from other simulation tools in that it combines workflow and GOMS models for assessment of correctness and performance of complex multi-users and multi-devices interactions.

3 Examples of UCAADS

This section briefly describes a smart medication dispenser, a smart storage pantry and a multi-user medication station to make the paper more self-contained. We use these devices and their operations in later sections for illustrative the purposes and capabilities of USE and as subjects of case studies.

3.1 Smart Medication Dispenser

A smart medication dispenser [3, 60] is designed for users who take medications over long periods of time without close professional supervision. In particular, it is designed to eliminate two common causes of administration error: misunderstanding of medication directions and inconvenience of rigid medication schedules. The dispenser schedules individual doses of the user's medications under its care based a machine readable *medication schedule specification (MSS)*. MSS is compiled from the user's prescriptions and directions of over the counter (OTC) drugs by the user's pharmacist.

To be concrete, we consider here the configuration of the dispenser shown in Figure 1. When a user comes to get new medication supplies, the pharmacist gives the user an updated MSS in a memory card together with medication containers. Each container holds one kind of medication, and the medication is identified by the RFID in the tag on the container. The user makes the dispenser ready to manage the medications along with existing ones by plugging in the memory card into the MSS port and the new containers in empty sockets, one at a time, in any sequence.

Figure 1 Smart dispenser

The dispenser reminds the user at the times when some dose should be taken. The user may or may not response promptly. When the user is late, the schedule may need to be adjusted. The work to ensure that the right doses of right medications are given to the user at the right time is done collaborative by the medication scheduler and the dispenser controller. Figure 2 shows an example of the communication between the scheduler and the controller. In this example, the user is supposed to take a 10 mg dose of insulin every 4 hours. If the user is tardy for more than 4 hours, the pending dose is cancelled and a double-size dose is scheduled. Furthermore, MSS specifies that the user's physician is to be notified if the user has not taken any dose for 10 hours or more.

Figure 2 shows what have taken place during part of a day. In Section 6, we will show how we simulate the interactions among the user, the dispenser controller and scheduler by using USE.

- The user has taken a dose of insulin promptly shortly after 9:00. A dose is scheduled at 13:00. At the time, the controller calls GetNextAction() to query for action. The action list returned by the scheduler includes turn on the local alarm (i.e., deliver reminder), start to monitor the PTD (Push-To-Dispense) button and prepare to help the user retrieve a 10 mg dose when the user pushes the button. After it queues the work items for these actions, the controller sets the NHST (Next Handshake Time) timer to expire at 17:00 and returns to wait,

while the worker threads process the work items. The threads are represented by wiggly lines in the left side of the figure.

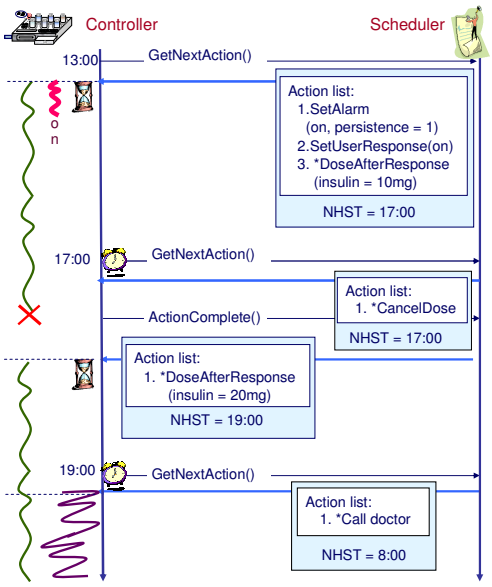


Figure 2 Scheduler and controller communication

• When the controller wakes up at 17:00 and calls GetNextAction(), the user still has not responded and the dose scheduled at 13:00 is still pending. The scheduler is aware of the fact because the controller has not yet reported the completion of DoseAfterResponse action. Since more than 4 hours has elapsed, the scheduler tells the controller to cancel the pending dose, while it adjusts the schedule according to the instruction from MSS.

• When the controller reports the completion of CancelDose, the scheduler requests that a 20 mg dose be given to the user when the user responds. The value of NHST returned by the scheduler this time is 19:00. By then, 10 hours will have been elapsed since the user took the latest dose of insulin.

• At 19:00, the user still has not come to push the PTD button. The scheduler requests that the controller calls the designated care taker to report the non-compliance event. The wide wiggly line on the left side of the figure represents the thread that logs the event and calls the care taker.

3.2 Smart Storage Pantry

A smart pantry [2] is for storage of non-perishable household supplies, such as detergent and shampoo. The pantry is smart because when the last unit of any kind of supply is removed from the pantry, the pantry automatically contacts a specified supplier, places an order and arranges payment on user's behalf to have replenishment delivered. This work requires the pantry to be able to identify the objects stored in it. We have built and experimented with pantries that use different technologies (i.e., RFID, digital camera and bar-code) for identifying objects and found that a BAC (bar-code) pantry has the best tradeoff between cost and usability.

A BAC pantry identifies objects in it by their bar-codes and sends the bar-codes of the objects in its orders to the supplier. Each shelf in such a pantry is partitioned into compartments. Associated with each compartment is a sensor, which allows the pantry controller to determine whether the compartment is empty or non empty. A restriction is that each compartment is used to hold only one kind of object.

A BAC pantry requires the user to scan the bar-code of the object when placing an object into an empty compartment. Load-pantry and remove-pantry are two major user operations. Load-pantry consists of user actions for placing objects into the pantry. Remove-pantry consists of user actions that remove objects from the pantry. We will return in Section 4 to describe details of these operations as illustrations of the UCAADS model.

3.3 Multi-User Medication Station

A medication station is a system of smart cabinets that are integrated by a server to provide storage and dispensing services in a patient ward. State-of-art medication stations (e.g., [61, 62]) operate in fully automated mode: When a user (normally, a nurse) comes to retrieve medications for a patient, the station opens automatically all the compartments (drawers) holding medications due to be administered to the patient at the time. Operating in this mode, a station can serve only one user at a time. The added burden on nurses to stand in line for retrieval of medications and to

adjust their work plans in order to minimize queuing time is a serious shortcoming. In contrast, multi-user medication station (MUMS) can also operate in semi-automatic mode [16]. In this mode, it allows multiple users to retrieve medications from the same station at the same time.

Figures 3 and 4 show a possible user scenario. As Figure 3 shows, each medication compartment (or drawer) in a MUMS has a label and a small LCD. The label has the name, bar-code and administration instruction of the medication inside. When a nurse comes to the station to retrieve medications, the LCD lights up and shows the names of the nurse and the patient and thus, helps the nurse locate the compartment.

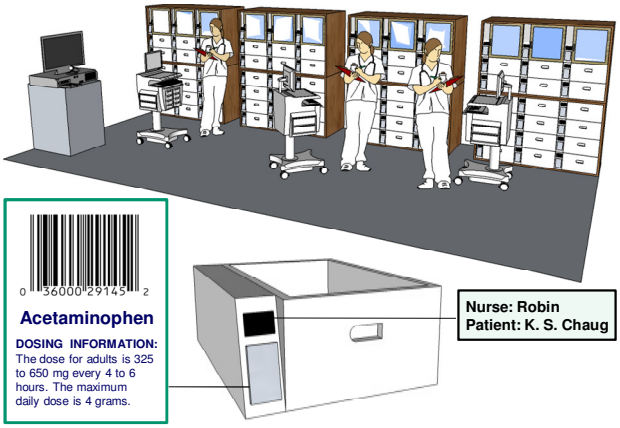


Figure 3 MUMS: Multi-User Medication station

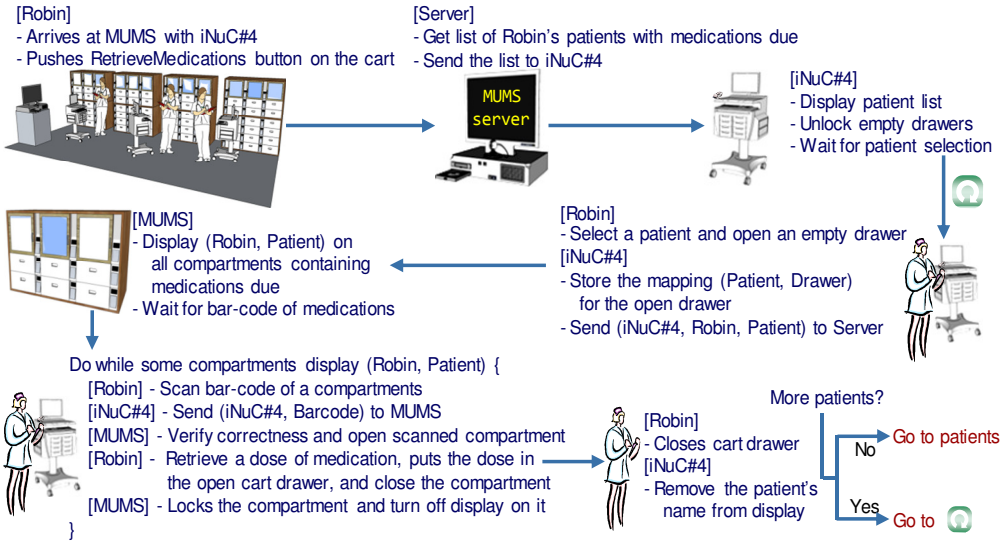


Figure 4 Bar-code controlled medication dispensing by MUMS

The flow diagram in Figure 4 illustrates the interactions between a user (say a nurse named Robin), MUMS and a mobile nursing cart during bar-code controlled medication dispensing. When Robin comes to work and signs on the MUMS server, the server retrieves Robin's patient list and her patients' prescriptions and medication schedules from the hospital database. The MUMS server sends a reminder via designated devices to Robin shortly before it is time for her to come to the station for retrieval of medications. We omit this part in Figure 4 to save spaceRobin responds to the reminder by logining a nursing cart (referred to as iNuC#4 in the figure) and bringing the cart to the station. After the station server authenticates Robin and discovers the ID of the cart, it sends to the cart the list of Robin's patients who are due to take medications. The list contains patient names and their IDs. The ID of each patient is the bar-code on the waistband of the patient. The monitor of her cart displays the patient list. Robin selects a patient at a time via the monitor. For each selected patient, she opens an empty drawer in the cart. The cart automatically associates the id of the selected patient with the location of the opened drawer. Once Robin finishes retrieving all the medications of the selected patient, she closes the opened cart drawer. The drawer can be opened again only by scanning the bar-code id on the patient's waistband.

After Robin selects a patient, the cart monitor displays the MUMS compartments where the medications of the patient are. At the same time, the MUMS compartments holding the medications display Robin's name and the patient name. To retrieve a dose of medication from one of these compartments, Robin uses the bar-code scanner attached to the cart to read the bar-code on the label of the compartment. The computer on the cart then sends the nurse's ID and the reading of the scanner to the MUMS server. After verification, the server unlocks the compartment, allowing Robin to retrieve a dose from it.

After retrieving a dose, Robin closes the MUMS compartment and then goes on to scan the bar-code on another compartment displaying her name, until a dose of every medication due to

be administered to the patient has been retrieved. Robin then closes the patient's drawer in the cart and moves on to retrieve medications of another patient, if any.

4 UCAADS Model

As stated earlier, an operational specification of a UCAADS describes the behavior of the device and the operations of the user (or users) in terms of one or more workflows. GOMS models elements are used to capture quantitatively the ability of the user in performing operations on the device. We use the model elements together to simulate the device and user as a whole for evaluation of device usability and prediction of user performance.

4.1 Workflow Elements of UCAADS Model

Specifically, a workflow is composed of elementary steps, called *activities*. Some of them are software activities, i.e., programs executed on CPU(s). Other activities are carried out by hardware components or by the user or users. The orders and conditions under which activities in a workflow are executed, the resources (e.g. software programs, hardware devices, users, etc.) needed for their execution, and interactions and communications among activities are specified either textually or by one or more workflow graphs. As stated earlier, we can define a workflow in terms of either the programming language C# [21] or the process definition language SISARL-XPDL [23]. In a workflow graph, each node represents an activity (or a state of the workflow). Each directed edge defines a *transition* between the activities (or states) represented by the source and sink nodes of the edge.

An essential component of USE is a workflow management system which is a middleware that provides a workflow manager and a workflow engine. The workflow manager schedules workflows and activities in them and have the workflow engine execute them. Again, USE uses either Microsoft .NET Workflow Foundation (WF) [21] or our own embedded workflow framework EMWF [23] for this purpose. Both workflow managers provide *built-in activities* as a part of the workflow engine. Built-in activities alter the timing, condition or flow path of the

execution of workflows, while activities provided by the developer usually do not. Our subsequent discussions assume that WF is used except where it is stated otherwise. Table 1 lists examples of built-in activities. The names of some of them (e.g., if-else and wait event) more or less tell what they do. A workflow can be a *sub-workflow* of a larger workflow or comprises *sub-workflows* and invokes them (i.e., called them asynchronously) or executes them (i.e., called them synchronously).

Table 1 Examples of build-in activities

Built-in Activities	
Start	Invoke workflow
Stop	Execute workflow
Repeat point	Invoke activity
While	Wait event
If else	Exception
Delay / Timeout	

There are two types of workflows, *sequential workflow* and *state machine workflow.* A sequential workflow consists of a fixed sequence of execution steps. The execution path may branch, or loop, etc. but has a workflow-wide starting point and ending point. A state machine workflow is event driven. A state machine workflow defines a set of states and possible transitions between states. Each state may have one or more activities that are executed prior to a transition to another state. The executions of all or most activities, and hence the transitions, are triggered by external events. We use both sequential and state machine workflows in operational specifications of devices and models of users and their actions.

For illustrative purpose, we go back to the load-pantry and remove-pantry operations of the smart storage pantry. Figure 5 shows a part of the operational specification that defines the load-pantry operation [2]. The user workflow in left side of the figure is sequential. It defines activities that model user actions. We will return to describe the incorporation of GOMS models with user workflows. The state machine workflow in the right side of the figure is the device

workflow. It specifies device behavior and services in the load-pantry process during which the user puts objects into the pantry.

A user starts load-pantry by pushing the LOAD button on the pantry. The button triggers the pantry to turn on the bar-code scanner and then returns to wait for the user to scan the bar-code of an object to be put into an empty compartment. After the user scans the bar-code and puts the object into the compartment, the storage pantry stores the association between the bar-code of the object and id of the compartment. The load-pantry process (i.e., the workflows) stops after the user puts all the objects into the pantry and the pantry timeouts waiting for bar-code. When a compartment becomes empty, the pantry inserts the bar-code associated with the compartment in the purchase order. It then deletes the association and thus frees the compartment for new supplies.

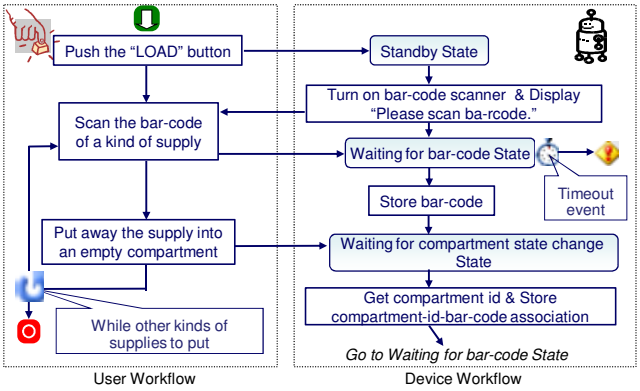


Figure 5 Load-pantry operation of the smart storage pantry

Figure 6 shows the workflow graph that specifies remove-pantry. When the last object is removed from a compartment (named Comp[k] in the figure), the pantry detects that the compartment is empty. It is possible that the user did not follow the load-pantry procedure as described, but simply put the objects in the compartment without scanning their bar-codes. In that case, the bar-code associated with the compartment is NULL. The pantry requests the user to scan the bar-code of the just removed object. If the user ignores the request, the pantry generates an error message to inform the user of its failure to reorder the object when timeout occurs. If the

pantry has a bar-code associated with the compartment or if the user has responded the request of scanning the bar-code of the removed object, the pantry generates and sends the order containing the bar-code to the specified supplier.

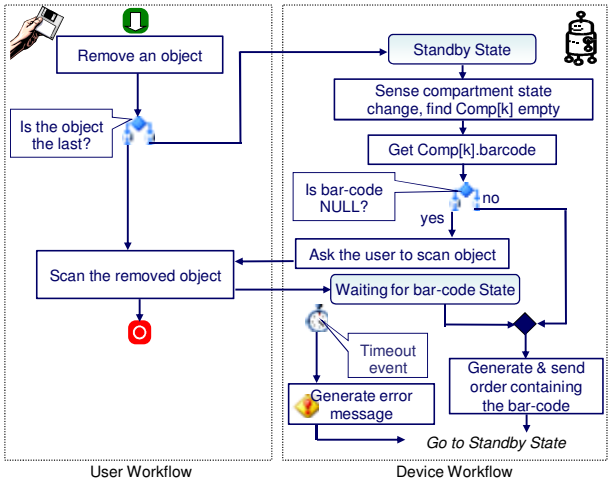


Figure 6 Remove-pantry operation of the smart storage pantry

4.2 GOMS Elements of UCAADS Model (GOMS Workflows and Activities)

Oftentimes, the developer also wants to know the lengths of time required to complete important operations, referred to as *response times*. For example, before we deploy MUMS in a ward, we will want to determine the maximum number of users the server should allow to retrieve medications concurrently. The response time of a semi-automatic operation depends not only on the user-device interaction, but also on the execution times of user actions. By incorporating GOMS models of human behavior with the workflow model, we can more precisely model user actions and estimate execution times of the actions.

As stated earlier, USE supports NGOMSL and CPM-GOMS variants of GOMS model. The hierarchical goal structure of NGOMSL resembles the structure of workflow (activity, composite activity, sub-workflow, and workflow). We can easily translate a NGOMSL analysis into executable and reusable workflows, sub-workflows and activities. As an example, Figure 7 shows a NGOMSL analysis of the user actions during the load-pantry operation. Here, the

analyst starts with the highest level goal "load pantry", and then breaks it into subgoals. Each subgoal is to be achieved by a specified method or methods and may be further broken into more detailed subgoals to be achieved by their own methods.

```
Method for goal: load pantry
  Step 1. Accomplish goal: push <LOAD> button.
  Step 2. Accomplish goal: scan the bar code.
  Step 3. Accomplish goal: put objects.
  Step 4. Return with goal accomplished.

Method for goal: push <LOAD> button.
  Step 1. Locate <LOAD> button on the screen.
  Step 2. Move hand to <LOAD> button.
  Step 3. Press <LOAD> button.
  Step 4. Return with goal accomplished.

Method for goal: scan a bar code.
  Step 1. Wait for response from the pantry.
  Step 2. Verify the message on the screen.
  Step 3. Accomplish goal: grab the bar code scanner.
  Step 4. Accomplish goal: grab the object.
  Step 5. Locate the bar-code on the object.
  Step 6. Click the scanner to scan.
  Step 7. Put down the bar code scanner.
  Step 8. Return with goal accomplished.

Method for goal: put objects.
  Step 1. Accomplish goal: choose an empty compartment.
  Step 2. Accomplish goal: put the objects into the compartment.
  Step 3. Return with goal accomplished.
```

Figure 7 Top-Level-Methods of the load-pantry operation in NGOMSL model

CPM-GOMS variant supports concurrent execution of human operators. In this sense, it is most compatible with workflows among four variants and more natural for modeling many UCAADS user actions. To incorporate CPM-GOMS model with workflows, we implement basic and frequently used human operators (e.g., perceptual, cognitive and motor operators) as *human-operator activities*. We implement the rules (e.g. Fitt's law [63] for moving a hand, or an empirical probability function for taking an object) and parameters (e.g., start and end positions of the mouse cursor) that govern the definitions of operators and calculations of execution times or learning times of users by parameterized functions executed by human-operator activities. Human-operator activities are components with which a developer can construct composite activities and sub-workflows that represent simple actions of the user (e.g., push a button).

To illustrate, Figure 8 shows a fragment of CPM-GOMS model of a user pressing the LOAD button on the pantry. Except for precedence constraints indicated by directed edges in the graph,

operators can execute in parallel. Figure 9 shows the equivalent workflow model fragment. By being equivalent, we mean that the workflow model "implements" the CPM-GOMS fragment and allows the model to run during simulation.

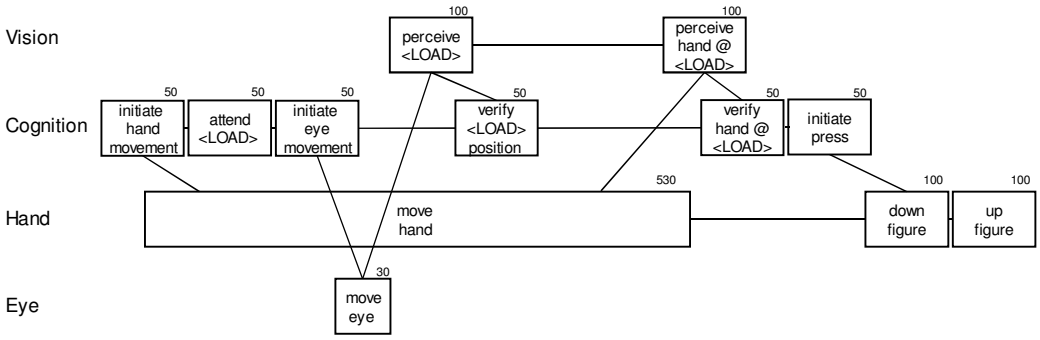


Figure 8 Push LOAD button in CPM-GOMS model

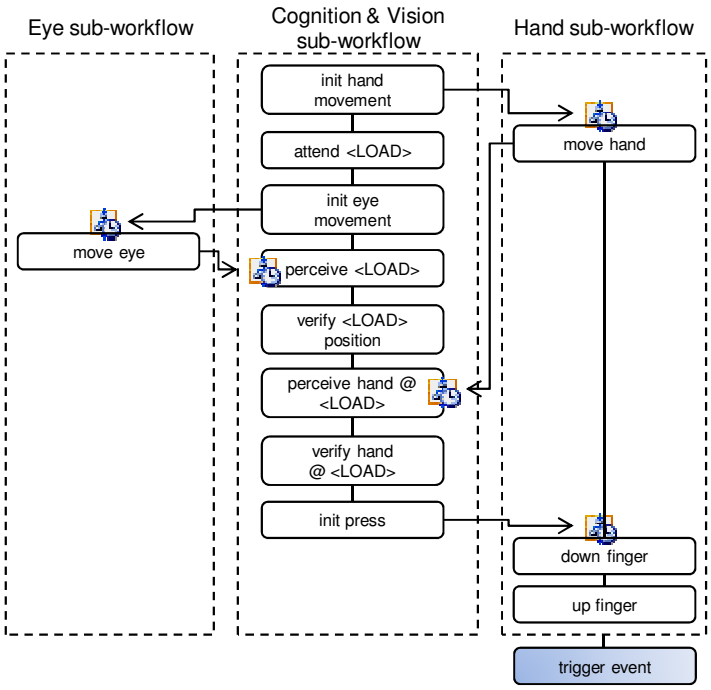


Figure 9 Push LOAD button CPM-GOMS template in workflow

For sake of simplicity, cognition and visual perception operators in the example are implemented by a single sub-workflow, called cognition & vision sub-workflow. The actions of eyes and hands are executed by eye sub-workflow and hand sub-workflow, respectively. Similarly, each activity in the sub-workflows has a function that estimates the execution time of user. The

sub-workflows synchronize via the wait event build-in activity. The down finger activity is enabled and executed by the workflow engine when the move hand activity completes and init press completed event occurs (i.e., the init press within the cognitive sub-workflow completes). In this way, we capture in workflows the precedence constraints in CPM-GOMS fragment.

5 UCAADS Simulation Environment

Figure 10 shows the structure of UCAADS simulation environment. Again, a developer specifies the operations of the device being evaluated by device workflow(s) and user actions by user workflow(s). The simulation environment takes them as input of a simulation experiment.

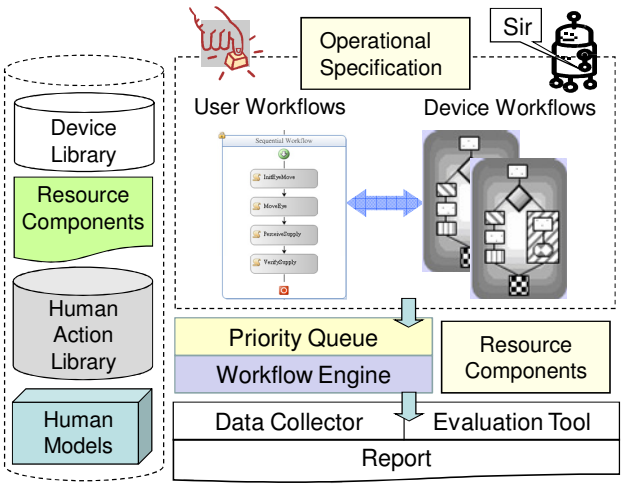


Figure 10 Architecture of UCAADS Simulation Environment (USE)

5.1 Local Service and Workflow Communication

USE local service provided by the simulation environment is a communicated channel between user workflows and device workflows in an operational specification. It is built on the custom data exchange services (called local services) supported by WF.

USE local service interface is defined as follows:

```
interface IUSELocalService {
  RegisterWorkflow(String identifier, Type wfType, Guid workflowInstance);
  OperateDevice(String deviceIdentifier, OperateDeviceEventArgs eventArgs);
  RespondToUser(String userIdentifier, RespondToUserEventArgs eventArgs);
  event EventHandler<OperateDeviceEventArgs> OperateDeviceEvent;
  event EventHandler<RespondToUserEventArgs> RespondToUserEvent; }
```

When a workflow is created at the start of a simulation, it calls RegisterWorkflow method to register itself with USE local service so that the local service can deliver events to it.

Figure 11 shows the use of OperateDevice method: A user workflow calls OperateDevice method to raise an OperateDeviceEvent event. The event carries information on a user action (e.g., click, push, open, etc.), the target of the action (e.g., scanner, LOAD button, drawer #1, etc.), and other information (e.g., bar-code, medication-id) associated with the action. USE local service delivers the event to the device workflow specified by the deviceIdentifer argument. A device workflow calls RespondToUser method to raise a RespondToUserEvent event to react to a user workflow. The event carries the output type of the response and the message of the response.

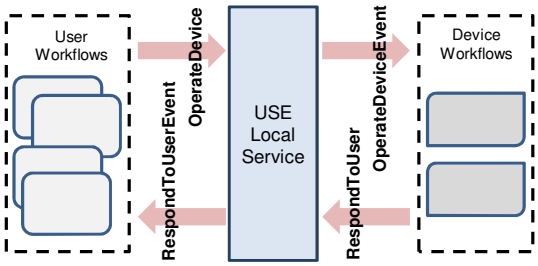


Figure 11 USE local service

5.2 Libraries

As Figure 10 shows, USE has four libraries: device library, human action library, resource components and human models. A developer can easily extend the libraries by putting new components into the libraries for use later. The device library contains reusable activities and workflows (e.g., load-pantry and remove-pantry device workflows in Figure 5 and 6) for constructing device workflows. Resource components include dynamically linked library (DLL) functions, executable and other types of building blocks required by workflows to carry out activities. As examples, many UCAADS (e.g., medication dispenser [3], iNuC [15], and MUMS [16]) have drawers. A sensor on each drawer is used to detect whether the drawer is empty or nonempty. In the model of such a device, one or more device workflows specify the actions to be taken when a drawer changes from empty state to non-empty state and vice versa. The resource

components required by the device workflows include emulators of sensor hardware and driver.

Human action library holds activities and workflows (e.g., human-operator activities and the push LOAD button CPM-GOMS workflow described in the previous section) from which developers can construct user workflows. The activities and workflows in this library are implemented as DLL functions.

A developer can put frequently used activities and workflows in the toolbox in the workflow editor provide by WF, and later drag them from the toolbox to build new user workflows. Figure 12 shows a part of the activities and workflows in the toolbox for constructing CPM-GOMS models. When an activity or workflow is dragged from the toolbox to the workflow editor, the toolbox pops a picture of CPM-GOMS PERT chart of the activity or workflow for the developer's information.

The forth library in USE is the human model library. As stated earlier, human models allow the developer to take account of factors that influence execution times of activities in user workflows. Examples of factors are skills of users, ages of users, and environments (e.g., dark room) etc. Some targeted users of UCAADS are elderly individuals and staff members of care-providing institution. We are collecting data to generate human models for these users and store the models in XML format.

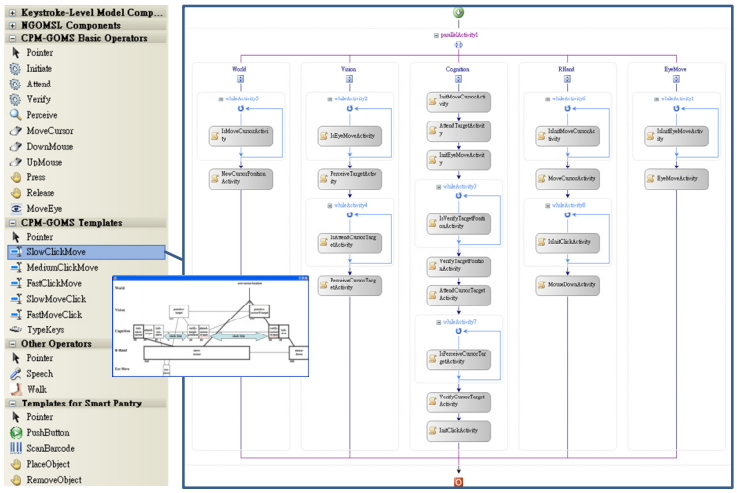


Figure 12 Human actions in toolbox

5.3 Priority Queues, Evaluation Tool and Data Collector

Most UCAADS have tasks that provide time-critical services. These tasks usually execute at different priorities in order to ensure their timely completion. Workflows simulating these tasks need to execute at the priorities of the tasks. To support priority-based design, USE provides a customized scheduling service rather than relying on the default scheduling service of the Microsoft WF. USE scheduling service manages a specified number of prioritized work queues. A developer can assign a priority to a workflow. During the execution of the operational specification containing the workflow, the workflow is inserted into the work queue of the assigned priority as a work item and is executed at the priority by a thread serving the work queue. The current USE scheduling service does not allow varying priorities within workflows (i.e., have activities in a workflow executed at different priorities). This limitation prevents the simulation of workflow-based devices to run on EMWF [23]. As a part of future work, we will extend the USE scheduling service to eliminate this limitation.

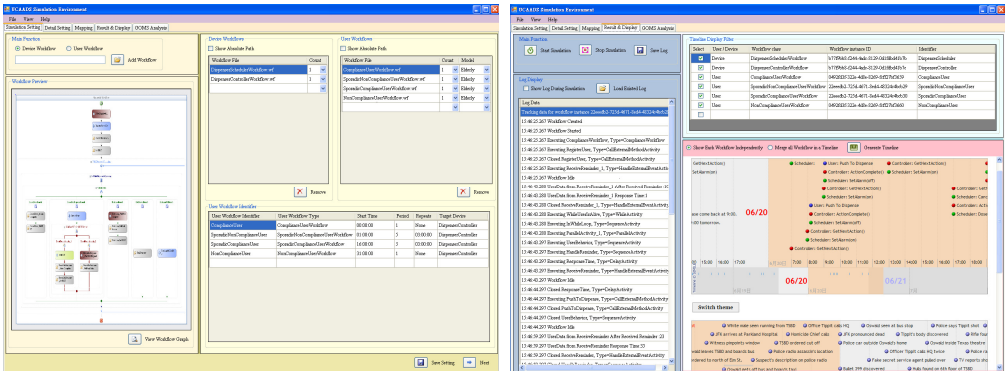
Oftentimes, human errors are in fact design errors of devices. When a user operates a device, he/she expects that the device to enter a certain state or executes certain service. Problems arise when the expectation and view of the user differs from the actual state and behavior of the device. This kind of discrepancy, sometimes called automation surprise, is known to cause serious accidents, errors and harm. To help the developer identify and eliminate user-device interaction that can cause this kind of problem, USE provides an evaluation tool with which the developer can set pre-conditions and post-conditions of individual activities. A pre-condition of an activity (or a sub-workflow) defines an initial state of the workflow immediately prior to the execution of the activity (or the sub-workflow). A post-condition defines a final state after the execution completes. Take the activity called “Scan the bar-code…” in the user workflow of Figure 5 as an example. A pre-condition and a post-condition of the sub-workflow are:

pre-condition: PantryController.state == AWAITS_BAR_CODE

post-condition: Scanner.register != NULL && Scanner.register == User.object.barcode

In other words, the user (or tester) thinks that the pantry controller should be in AWAITS_BAR_CODE state before the "Scan the bar-code…" is executed. After the sub-workflow is executed, the register of the bar-code scanner should stores a valid bar-code value and the value is the bar-code of an object to be put into the pantry. The conditions are checked at runtime, and execution of the workflow pauses for attention when any condition is violated. By setting pre- and post-conditions, a developer can observe whether the workflows execute as expected and whether the simulated device behaves as expected.

Finally, USE data collector logs details on transitions of activities, and changes in variables and conditions during each simulation run. It also can capture the events specified by the developer and interactions among user workflows and device workflows. The data are stored in a database for off-line analysis and display. Figure 13 shows the GUI of USE for setting up the simulation and showing simulation results. The screenshot in part (a) shows a GUI tab for a developer to load and remove workflows and complete initial settings. The screenshot in part (b) shows another GUI tab for displaying the logs and timelines of simulation data generated from logged data by an open source timeline widget [64] embedded in USE.



(a) (b)

Figure 13 GUI of USE

5.4 Development Process of UCAADS

We conclude this section by describing how USE can assist developers during different

phases of the development process. To be concrete, we divide the developers of a new UCAADS into three teams: *test team*, *system team* and *user interface team*. The test team is responsible for building user workflows used to test all kinds of user scenarios. The system team is responsible for implementing device workflows and functions of the device/system. If the device/system has a GUI, the user interface team is responsible for the design and implementation of the GUI.

In the first phase, developers work cooperatively to define the operational specification of the new device. The system team uses state machine and sequential workflows to depict the skeleton of device behaviors such as states transitions and activities for handling events trigged by user actions. The test team creates all kinds of user workflows (e.g., for normal user, exceptional user, arbitrary user, etc.) for testing. Testing in this phase aims to explore all the possible user scenarios and make sure that the device handles these scenarios in correct manner and transits to the correct state. In this step, device workflows and user workflows interact through USE local service described above. In the meantime, the user interface team design and implement another local service, called GUI local service, which is for communication between the GUI and device workflows. Figure 14(a) sketches the configuration used in this phase.

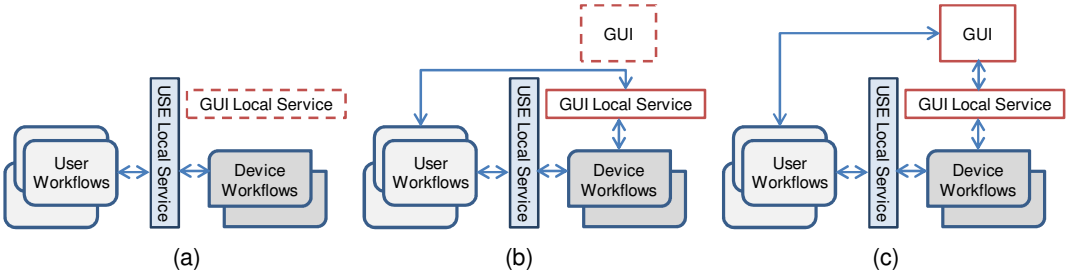


Figure 14 Development process of UCAADS

In the second phase of the development process, the system team starts to implement functions of the device in this phase. Resource components can be used to carry out parts of these functions. While the GUI is still under construction and is unavailable for testing purpose, user workflows can be used to trigger events on GUI local service. These events simulate events from the GUI triggered by user actions that operate the GUI. Similarly, user workflows can be

used to trigger events on USE local service to simulate user actions that are not related GUI (e.g., take drugs, open a drawer). Figure 14(b) sketches the configuration used for testing in this phase.

In the final phase of the development process, the new device is almost completed. The test team can test the GUI, GUI local service and device workflows as a whole. In particular, once the GUI is completed, user workflows can be used to simulate GUI operations by operating the GUI directly. Figure 14(c) gives a sketch of this phase.

6 Case Studies

We have used USE to assess the design of the UCAADS described in Section 3. We ran device workflows and user workflows of these devices in order to identify defects of their designs: Serious defects exhibit themselves by the inconsistency between pre/post conditions and device states captured by the evaluation tool. The simulation experiments also gave us estimated execution times of user operations.

6.1 Simulation of Smart Medication Dispenser

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. 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 non-compliance events.

Figure 15 shows parts of the operational specification of the dispenser. It is a part of the input provided to USE in a simulation experiment. 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 takes to complete elementary steps in user activities in the user workflow. The most important parameter is the probability distribution of the response time (i.e., the time between when a reminder is sent to the time when the user comes to push PDT button). During each simulation run, timeout length of the response timer is set to a sample value from this distribution. In

addition, 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 [65].

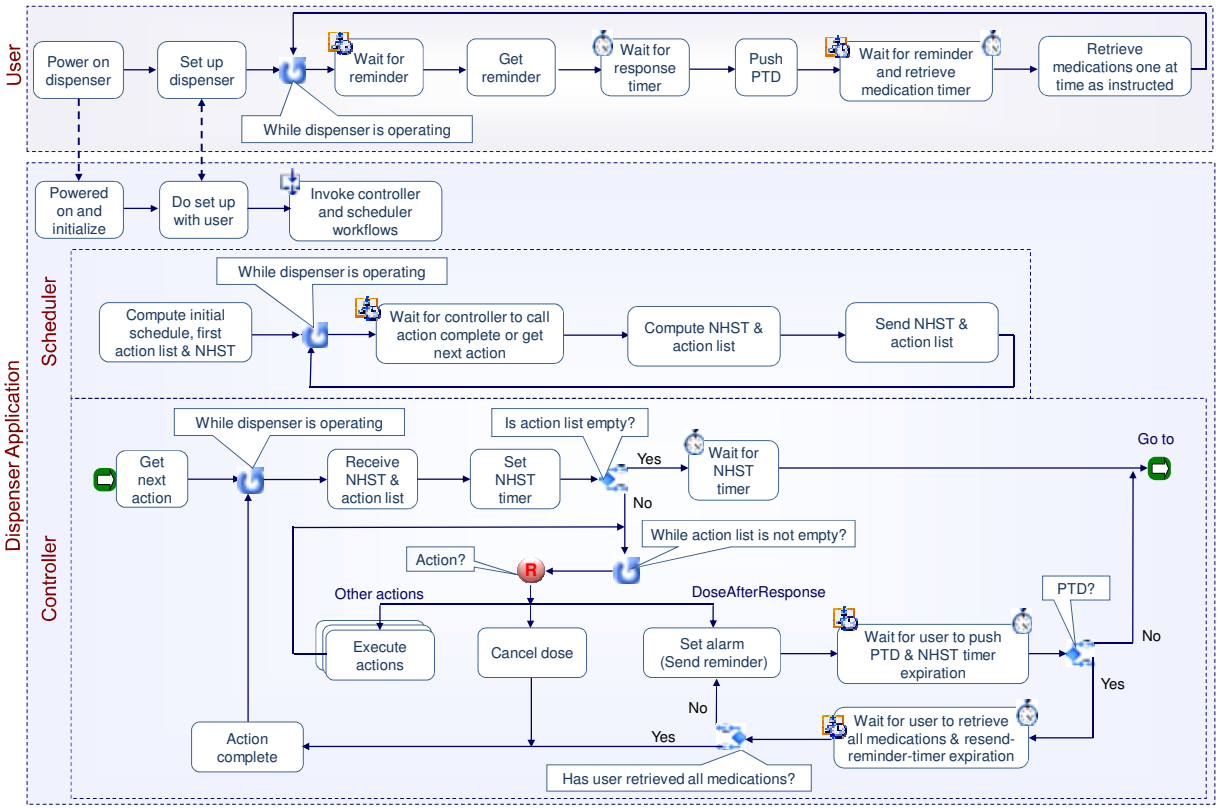
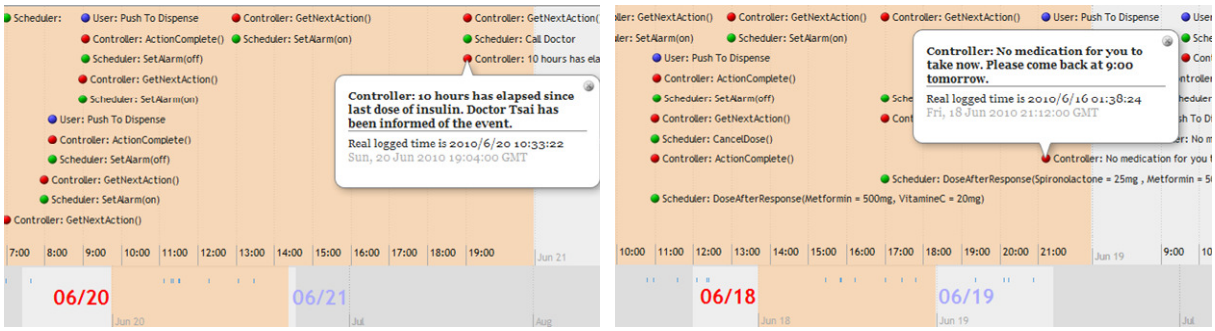


Figure 15 Parts of the operational specification of the dispenser and user actions

During simulation, USE captures and logs events in interactions of the user, and the controller and scheduler of the dispenser. As an 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 correctly instructed the user to retrieve each 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, Figure 16 shows fragments of event traces captured in two simulation runs. 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. Both simulation traces are 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. The trace in Figure 16 (a) shows that the dispenser correctly handles the non-compliance event described in Figure 2. The trace in Figure 16 (b) shows an event raised by the controller in response to the user pushing the PTD button at times when no medication is due.



(a) (b)

Figure 16 Examples of simulation data

6.2 Simulation of Smart Storage Pantry

The process the smart pantry takes to acquire compartment id and bar-code associations is error prone. A busy user may dump new supplies in the pantry without scanning their bar-codes. Multiple users may put away supplies and remove supplies at the same time. We cannot strictly restrict user-pantry interaction patterns but must be sure that the pantry works satisfactorily regardless. For this reason, an objective of simulations of the smart pantry is to determine whether and how inconsistency between bar-codes of objects recorded by the pantry and actual bar-codes of objects can occur. Such inconsistencies cause the pantry to order wrong supplies for the user, which is deemed unacceptable. We started our experiment by simulating the scenario where a single user operates load-pantry and remove-pantry, one operation at a time. We then repeated the experiment numerous times, each time with multiple users doing load-pantry and remove-pantry simultaneously.

As we expected and hoped, the simulation results expose at least two design errors when the pantry is operated by more than one user at the same time. This scenario is highly likely in most households. We were able to reconstruct from the recorded event logs detailed use scenarios that led to the errors reported by USE. As example, the scenario in Figure 17 explains how the load-remove error occurred. Before diving into details of the scenario, we note that for sake of usability, the pantry allows objects to be placed into empty compartments without first being scanned. Consequently, there may not be a bar-code associated with a compartment when the last object in it is removed. When this happens, the pantry tries to acquire the missing bar-code before the user removing the last object goes away. This is why remove-pantry workflow is executed at a higher priority than the interactive load-pantry workflow process.

User 3: ***Push "Load " Button .***
Pantry: ***Turns on bar-code scanner and expects the user to scan each object and then put the object away.***
User 3: ***Scan bar-code of a six pack of mineral water.***
Pantry: ***You just scanned an object.***
User 4: ***Remove the last bottle of coke in compartment 4.***
Pantry: ***You just remove the last unit of the unknown object. Please scan bar-code for re-order.***
User 4: ***Scan bar-code of coke.***
Pantry: ***Re-order?***
User 4: ***Push "Yes" Button.***
User 4: ***Walk away.***
User 3: ***Put a six pack of mineral water into compartment 3.***
Pantry: ***You have just put an object in compartment 3.***
User 3: ***Walk away.***

Figure 17 Scenario of load-remove error

Returning to Figure 17, we see that just after user 3 scanned the bar-code of a pack of mineral water, user 4 removed the last bottle of Coca-Cola from a compartment for which the pantry had no bar-code. By preempting the load-panty workflow immediately to capture the missing bar-code without first saving the bar-code captured in the load-pantry process, the pantry mistook the bar-code of Coca-cola, which user 4 scanned in the remove-pantry process, for bar-code of mineral water, which user 3 put into compartment 3. In this case, there are many ways to fix the problem, (e.g. save the bar-code captured by load-pantry before preempt the

workflow, or requesting the user to rescan the object put into a compartment immediately after load-pantry resumes, etc.) once we are aware of the problem.

6.3 Simulation of Multi-User Medication Station

In an experiment to evaluate the multi-use medication station (MUMS) described in Section 3.3, we simulated different numbers of users (from 2 to 10 nurses) retrieving medications from a multi-user medication station in the same time. Similar to the experiment on the smart storage pantry, we first determined the correctness of the bar-code controlled dispensing process. The mobile nursing cart and the MUMS server operations and their interactions with each user during this process are described by Figure 4. By being correct, we mean that every user (a nurse) gets correct medication(s) from MUMS and puts the medications in an associated drawer of the cart correctly identified by the cart and MUMS server for each patient. Fortunately, in the simulation runs we have done to date we did not discover any inconsistency and malfunction caused by user-device interactions. Even though the user-device interaction shown by Figure 4 looks complicated, it actually has less chance for errors. An analysis of the recorded logs shows that, while the MUMS server allows multiple users to access and operate multiple compartments, it correctly allowed only one user at a time to operate any compartment. Moreover, unlike users of the smart pantry, who share resources (e.g. bar-code scanner), each of the MUMS users has his/her own cart and bar-code scanner.

To access the responsiveness of MUMS, this purpose, we simulated the scenarios where multiple nurses are using a medication station to get medications. Each nurse does the retrieval in order given by his/her patient list, and waits for a MUMS compartment if the compartment holding the next medication he/she is going to retrieve was being used by another nurse.

Table 2 lists the parameters of the simulation. Usually some medications in the MUMS of a hospital ward are frequently prescribed. We use a probability distribution (e.g. even distribution or Zipf's law [66]) to govern the selection of medications stored in the station. By adjusting the

distribution, we can fit the medication access patterns to different departments in a hospital. The number (138) of compartments of the MUMS was suggested by National Taiwan University Hospital, and the use pattern is based on actual data on use frequencies of medications provided by the hospital. The estimated times of user actions while operating the GUI on the nursing cart are modeled by CPM-GOMS. The amounts of time of other user actions (e.g., open a compartment, walk a distance, etc.) were obtained by measuring the time taken by ten people.

Table 2 Parameters of the MUMS Simulation

Number of patients for each nurse	6
Number of medications for each patient	6~14
Medications use pattern	Zipf's law
Drawers of MUMS	138
Operations on Basic Nursing Cart	CPM-GOMS
Other Operations	Measured
Machine Response Time	0.1s~1s

Figure 18 shows the average waiting time and average retrieval time taken by a nurse to retrieval all the medications of a patient as functions of the number of nurses using the MUMS at the same time. The former is the average amount of time wasted waiting for access to some compartments holding medications to be retrieved. The average retrieval time is the average amount of time taken to retrieve all medications of a patient without any waiting. It is unacceptable when the average waiting time is a significant portion of the average retrieval time. According to this data, we suggest that the MUMS server schedule at most three or four nurses to uses the MUMS at the same time.

7 Summary and Future Works

We described in this paper how the combination of workflow and GOMS models provides a flexible way to define device operations and user actions for the purpose of evaluating UCAADS and their user-device interactions. USE provides libraries of reusable model components for construction operational specifications and supports prioritized executing of workflows.

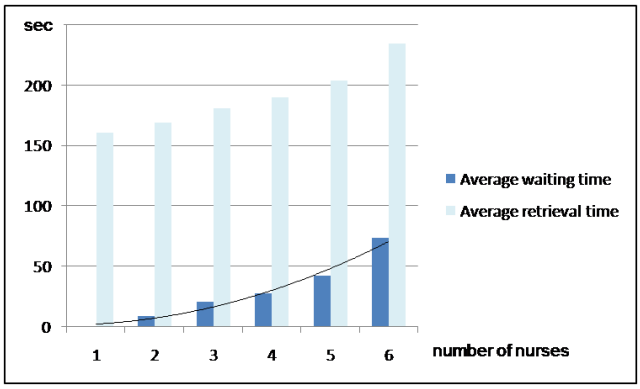


Figure 18 Average waiting time of a nurse finish retrieval of all medications

The simulation results can help us in two aspects throughout the three phases of UCAADS development. One is refinements of the device design. In simulation experiments on the dispenser, we examine the protocol between the controller and the scheduler of the dispenser and make sure the dispenser handle all non-regular compliance events. In experiments on the pantry, we found that the smart pantry may fail and order wrong replenishment when the pantry is operated by multiple users. By examining the recorded event sequence, we were able to determine the cause of the bug and fix it. In the experiment involving MUMS, we can improve the usability of the UI of the mobile nursing cart and simplify the process of medication retrieval. Also, we can implement a scheduler to schedule the nurses in order to minimize their contentions for MUMS compartments when they come to retrieve medications from MUMS at the same time. Another is giving suggestions of usages of UCAADS.

There are many works remain to be done. We want to design a script language and exploit current automation GOMS tools to automatically generate user workflows from the human action library. We will provide USE the capability of automatically generating GOMS models by simply demonstrating tasks with a graphic user interface that implemented in Windows Form and Windows Presentation Foundation in the future.

We plan to improve the evaluation and analysis tool to reduce the human efforts in finding complicated error scenarios. We also need to improve the GUI and visualization tools for setting

up USE and presenting simulation data. The experiments in this paper did not adopt human models. In the future, we will incorporate human models as parameters of user workflows to get user performance for different types of users.

Acknowledgement

This work is supported by Taiwan Academia Sinica Thematic Project SISARL, Sensor Information Systems for Active Retirees and Assisted Living. The authors wish to thank T. S. Chou and Y. C. Huang for their contribution on EMWF.

References

[1] Sensor Information Systems for Active Retirees and Assisted Living, http://www.sisarl.org/

[2] C. F. Hsu, Y. H. Liao, P. C. Hsiu, Y. S. Lin, C. S. Shih, T. W. Kuo, and J. W. S. Liu, "Smart pantries for homes," in *Proceedings of IEEE SMC,* October 2006.

[3] 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 *Proceedings of IEEE Real-Time Systems Symposium,* December 2006

[4] Y. Hsu, C. E. Chiang, Y. H. Chien, H. W. Tseng, A. C. Pang, T. W. Kuo, and K. H. Chiang, "Walker's buddy: an ultrasonic dangerous terrain detection system," in Proceedings of IEEE SMC, October 2006.

[5] T. S. Chou and J. W. S. Liu, "Design and Implementation of RFID-Based Object Locators," in *Proceedings of IEEE International Conference on RFID Technology,* March 2007.

[6] iRobot Home Robots, http://www.irobot.com/

[7] Forizzi, J. and C. DiSalvo, "Service robots in domestic environment: a study of Roomba vacuum in the home," in *Proceedings of ACM/IEEE International Conference on HRI,* March 2006.

[8] Kulyukin, V. A. and C. Gharpure, "Ergonomics-for-one in a robot shopping cart for the blind," in *Proceedings of ACM/IEEE International Conference on HRI,* March 2006

[9] Kaneshige, Y., M. Nihei, and M. G. Fujie, "Development of new mobility assistive robot for elderly people with body functional control," in *Proceedings of IEEE/RAS-EMBS,* February 2006.

[10] Lin, C. H., Y. Q. Wang and K. T. Song, "Personal assistant robot," in *Proceedings of IEEE International Conference on Mechatronics,* July 2005.

[11] Mataric, M. J., J. Eriksson, D. J. Feil-Seifer, C. J. Winstein, "Socially assistive robotics for

post-stroke rehabilitation," in *Journal of Neuroengineering and Rehabilitation,* Vol. 4, No. 5, 2007

[12] Gockley R., and M. J. Mataric, "Encouraging physical therapy compliance with hand-off mobile robot," in *Proceedings of ACM/IEEE International Conference on HRI,* March 2006.

[13] Thrun, S., "Toward a framework for human-robot interaction," in *Human-Computer Interaction,* Vol. 19, 2004.

[14] Fong, T., I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," in *Robotics and Autonomous Systems*, Vol. 42, 2003.

[15] P. H. Tsai, Y. T. Chuang, T. S. Chou, C. S. Shih and J. W. S. Liu, "iNuC: An Intelligent Mobile Medication Cart," in *Proceedings of the 2nd International Conference on Biomedical Engineering and Informatics*, October 2009.

[16] J. W. S. Liu, C. S. Shih, C. T. Tan and Vincent J. S. Wu, "MeMDAS: Medication Management, Dispensing and Administration System," in *International Mobile Health (mHealth) Workshop,* July, 2010.

[17] J. W. S. Liu, C. S. Shih, T. W. Kuo, S. Y. Chang, Y. F. Lu and M. K. Ouyang, "Flexible User-Centric Automation and Assistive Devices," in *Proceedings of Workshop on Adaptable and Reconfigurable Embedded Systems*, April 2008.

[18] Workflow definition, http://en.wikipedia.org/wiki/Workflow

[19] John, B. E. and Kieras, D. E., "The GOMS family of user interface analysis techniques: comparison and contrast," in *ACM Transactions on Computer-Human Interaction,* Volume 3, Issue 4, December 1996.

[20] Card, S. K., Moran, T.P., and Newell, A. (1983). T*he Psychology of Human Computer Interaction.* Lawrence Erlbaum Associates. ISBN 0-89859-859-1.

[21] Windows Workflow Foundation: http://msdn2.microsoft.com/en-us/netframework/aa663328.aspx

[22] BPEL (Business Process Execution Language), http://en.wikipedia.org/wiki/BPEL

[23] T.S. Chou, S.Y. Chang, Y.F. Lu, Y.C. Wang, M.K. Ouyang, C.S. Shih, T.W. Kuo, J.S. Hu, and J.W.S. Liu, "EMWF for Flexible Automation and Assistive Devices," in *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium* (RTAS), San Francisco, USA, April 13-16, 2009.

[24] WfMC: Workflow Management Coalition, http://www.wfmc.org/

[25] XPDL (XML Process Definition Language), http://www.wfmc.org/xpdl.html

[26] Pajunen, L. and S. Chande, "Developing workflow engine for mobile devices," in *Proceedings of IEEE International Enterprise Distributed Object Computing Conference,* 2007.

[27] Hackmann, G., M. Haitjema, C. Gill, and G. C. Roman, "Silver: A BPEL workflow process execution engine for mobile devices," in *Conference of Conference on Service Oriented Computing*, ICSOC 2006.

[28] Jing, J., K. Huff, B. Hurwitz, H. Sinha, B. Robinson, and M. Feblowitz, "WHAM: supporting mobile workforce and applications in workflow environments," in *Proceedings of the 10th IEEE Workshop on Research Issues in Data Engineering,* February 2000.

[29] BPMN (Business Process Modeling Notation), http://www.bpmn.org/

[30] YAWL (Yet Another Workflow Language), http://www.yawl-system.com/

[31] Open Source Java XPDL editor, http://www.enhydra.org/workflow/jawe/index.html

[32] WfMOpen, http://wfmopen.sourceforge.net/

[33] ProcessMaker: Open Source BPM and Workflow, http://www.processmaker.com/

[34] ActiveBPEL, http://www.activevos.com/community-open-source.php

[35] Bonita, http://www.bonitasoft.com/

[36] John, B. E. (1995) Why GOMS? interactions, vol. 2, no. 4. pp. 80-89.

[37] John, B. E. and Kieras, D. E., "Using GOMS for User Interface Design and Evaluation: Which Technique?," in *ACM Transactions on Computer-Human Interaction,* Volume 3, Issue 4, December 1996.

[38] S. K. Card, T. P. Moran, and A. Newell, "The keystroke-level model for user performance time with interactive systems," in *Communications of the ACM,* 23(7), 396-410, 1980.

[39] Kieras, D. E. (1997). *A guide to GOMS model usability evaluation using NGOMSL.* M. Helander, T. Landauer, and P. Prabhu (Eds.), Handbook of human-computer interaction. (Second Edition). Amsterdam: North-Holland. 733-766.

[40] Bonnie E. John, "Extension of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information," in *Proceedings of the 1990 Conference on Human Factors in Computing Systems,* Seattle, April, 1990.

[41] Lu Luo, and Bonnie E. John, "Predicting task execution time on handheld devices using the keystroke-level model," in *Conference on Human Factors in Computing Systems* (CHI '05) extended abstracts on Human factors in computing systems, Portland, OR, April 2005.

[42] John, B. E. & Suzuki, S. (2009) Toward Cognitive Modeling for Predicting Usability. Proceedings of HCI International 2009 (19-24 July 09, San Diego, CA).

[43] J. L. Drury, J. Scholtz, and D. Kieras, "Adapting GOMS to model human-robot interaction," in *Proceedings of*

the ACM/IEEE international conference on Human-robot interaction, Arlington, Virginia, March USA , 2007.

[44] Beard, David V., Smith, Dana K. & Denelsbeck, Kevin M., Quick and Dirty GOMS: A Case Study of Computed Tomography, *Human-Computer Interaction*, 11 (2) p.157-180.

[45] Williams, K. E. (2005). Computer-aided GOMS: A description and evaluation of a tool that integrates existing research for modeling human-computer interaction. International Journal of Human-Computer Interaction, 18, 39–58.

[46] Kieras, D.E., Wood, S.D., Abotel K. and Hornof, A. GLEAN: a computer-based tool for rapid GOMS model usability evaluation of user interface designs. In *Proc. of the 8th annual ACM symposium on User interface and software technology 1995*.

[47] Hudson, S. E., John, B. E., Knudsen, K., and Byrne, M. D. A tool for creating predictive performance models from user interface demonstrations. UIST'99: Proceedings of the ACM Symposium on User Interface Software and Technology, CHI Letters 1(1), 93-102.

[48] John, B., Vera, A., Matessa, M., Freed, M., and Remington, R. Automating CPM-GOMS. CHI 2002, ACM Conference on Human Factors in Computing Systems, CHI Letters 4(1), 147-154.

[49] Patton, E. W., Gray, W. D., & Schoelles, M. J. (2009). SANLab-CM - The Stochastic Activity Networking Laboratory for Cognitive Modeling. Proceedings of the 53rd Human Factors and Ergonomics Society Conference (Oct 18-23), San Antonio, TX.

[50] CogTool, http://cogtool.hcii.cs.cmu.edu/

[51] Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., Qin, Y. An Integrated Theory of the Mind. Psychological Review, 111(4) 1036--1060 (2004).

[52] Windows Form and Windows Presentation Foundation, http://windowsclient.net/

[53] The ns-3 Network Simulator, http://www.nsnam.org/

[54] SENSE: Sensor Network Simulator and Emulator, http://www.ita.cs.rpi.edu/sense/index.html

[55] SkyEye, http://skyeye.sourceforge.net/index.shtml

[56] MobileSim, http://robots.mobilerobots.com/wiki/MobileSim

[57] Webots, http://www.cyberbotics.com/

[58] Hartmann, B., Klemmer, S.R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., Gee, J. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of UIST 2006,* October 2006.

[59] Björn Hartmann , Loren Yu , Abel Allison , Yeonsoo Yang , Scott R. Klemmer, Design As Exploration:

Creating Interface Alternatives through Parallel Authoring and Runtime Tuning. In *Proceedings of UIST 2008.*

[60] Tsai, P. H., C. Y. Yu, W. Y. Wang, J. K. Zao, H. C. Yeh, C. S. Shih, and J. W. S. Liu, "iMAT: Intelligent Medication Administration Tools," To Appear in Proceedings of IEEE Healthcom, July 2010.

[61] Pyxis Medstation, http://cardinal.com/us/en/providers/products/pyxis/brochure/MS3500_spec_details_PLA .pdf

[62] Rx Showcase, http://www.rxinsider.com/prescription_dispensing_automation.htm

[63] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. Journal of Experimental Psychology, volume 47, number 6, June 1954, pp. 381-391.

[64] "SIMILE Widgets, Free, Opens-Source Data Visualization Web Widgets and More", http://www.simile-widgets.org/

[65] Tsai, P. H., C. S. Shih, and J. W. S. Liu, "Algorithms for scheduling multiple interacting medications," Foundations of Computing and Decision Sciences, Vol. 34, No. 4, 2009.

[66] Zipf's Law. http://en.wikipedia.org/wiki/Zipf's_law.