TR-IIS-11-002

Workflow Architecture for Model-Based Development of User-Centric Automation and Assistive Devices

T. Y. Chen, Y. C. Huang, C. S. Shih, T. W. Kuo and J. W. S. Liu

Mar. 24, 2011 || Technical Report No. TR-IIS-11-002
http://www.iis.sinica.edu.tw/page/library/TechReport/tr2011/tr11.html

To be presented at Workshop on Architectures for Cyber-Physical Systems, CPSWeek, April 2011

Workflow Architecture for Model-Based Development of User-Centric Automation and Assistive Devices

T. Y. Chen
Department of Computer Science
National Tsing-Hua University
Hsinchu, Taiwan
yen@iis.sinica.edu.tw

C. S. Shih and T. W. Kuo
Dept. of Comp. Sc. & Info. Eng.
National Taiwan University
Taipei, Taiwan
{cshih, ktw}@csie.ntu.edu.tw

Y. C. Huang and J. W. S. Liu
Institute of Information Science
Academia Sinica
Taipei, Taiwan
{ginger, janeliu}@iis.sinica.edu.tw

***Abstract*—This paper describes an on-going effort in building user-centric automation and assistive devices from workflows. Advantages of this approach include that workflow-based devices and systems are naturally componentized and easily configurable. Models of a new device, user actions and users defined in terms of activities and workflows are executable. We can use them to simulate the device and its interactions with the user for requirement capture and design purposes. Later on in the development process, programs and other resources required by the device workflows become available. By using them, the workflow model of the device becomes its implementation. The workflows used to model user actions define the use scenarios and scripts for testing and evaluation.**

Keywords-workflow-based model and design, user models executable behavior specification

I. INTRODUCTION

This paper provides an overview of our work [1-4] on the workflow approach to model, design, implement and evaluate UCAADS (user-centric automation and assistive devices and systems). Examples of UCAADS can be found in [5-14]. Some of these devices aim to improve the quality of life and self-reliance of their users, including elderly or functionally limited individuals. Other UCAADS are automation tools designed to help care-providing institutions improve their quality of care and reduce the cost of care delivery.

UCAADS share two important requirements: They must be flexible and safe to use. By a device or system being *flexible*, we mean that it can be easily configured and customized to rely on different infrastructures, support different processes, enforce different rules and policies, suite difference users and cater to each individual user's needs and preferences. By it being *safe*, we mean that it never does any harm even when misused, and critical erroneous operations are recoverable.

A. *Workflow Architecture for Flexibility*

Flexibility is one of the factors that motivated us to build UCAADS on workflow architecture. The basic building blocks of a workflow-based application are called *activities*. A *software activity* is done by executing a software procedure on a CPU. An *external activity* may be an operation by a hardware device, a network, and so on. Activities are composed into module-level components called *workflows*. The order and conditions under which activities in a workflow are executed and the resources needed for their execution are defined by the developer of the workflow. The definition can be in terms of a programming language (e.g., C# in [15]), a process definition language (e.g., WfMC standard XPDL, XML Process Definition Language [16, 17]), or an execution language (e.g., BPEL, Business Process Execution Language [18]). Workflows can also be defined graphically [11, 19]: In a workflow graph, nodes represent activities in (or states of) workflows and directed edges between nodes represent transitions between activities (or states).

A key component of a platform for workflow-based applications is a workflow manager/engine. The middleware manages the life cycles of workflows at runtime, allocates resources to their activities, sequences and schedules the activities in them, and supports their communication and interactions. It also provides and executes built-in activities (e.g., start, stop, wait-for, if-else, while) that control the flow and timing of execution paths of workflows, in addition to executing software activities. In this way, the workflow engine enables the control and information flows between components to be separated from the actual execution of code of the underlying components. This separation is the primary reason that components in a workflow-based device can be easily rearrange and changed even at runtime and the resultant device is easily configurable and customizable.

Today, there are a wide spectrum of matured engines and tools for defining, building and executing workflows (e.g., [15-22]). They are primarily for applications that automate business processes on enterprise computers and mobile devices, however. EMWF (Embedded Workflow Framework) [3] to be presented later in the paper is a workflow management system designed specifically for UCAADS and embedded devices in general. Typical UCAADS are not as severely power and size constrained as cell phones and other mobile devices and do not have challengingly stringent timing requirements. We take advantage of these characteristics of UCAADS in the design of the applications and the middleware to tradeoff between flexibility and other figures of merit, such as memory footprint and runtime overhead.

This work was partially supported by Academia Sinica Project SISARL.

B. *Workflows as Device and User Models*

While working on the design and implementation of workflow-based UCAADS, we came to appreciate the merit of using workflows as models and a basis of tools for ensuring that the devices are safe. Within the framework of UCAADS model [1, 4], we represent jobs and tasks done by the device as software and external activities and workflows (called *device activities* and *workflows*). Many UCAADS are semi-automatic. We model actions taken and tasks done by human user(s) by *user activities* and *workflows* and device-user interactions by services for workflow to workflow communication. We also use workflow definitions of GOMS (Goals, Objectives, Methods and Selections) and MHP (Model Human Processor) [23, 24] model elements to characterize human users with different attributes and skills.

Throughout the development process, a device or system is defined by its *behavior specification* (called operational specification in [1]) and *resource specification*. The behavior specification is the workflow model of the device and, hence, is defined in terms of definitions of device workflows. The definitions also specify the resources (e.g., hardware and firmware devices, sensors and actuators, dynamically linked library functions and executables, and human users) required by each activity. In the design phase before the actual resources are available, the workflow definitions call for virtual resources (i.e., device simulators/emulators, dummy code, user models, etc.) as resource components. As the development process progresses, the virtual resources are replaced by physical devices and programs, and the device workflows modeling module-level components become implementation of the components. User workflows provide use scenarios and scripts for testing purposes. The resource specification of a device describes the dependencies and compatibilities between versions of resources used by activities and workflows during the development process. UCAADS resource specification are defines of a nesC-like [24] resource description language. Details on the language can be found in [1, 25].

An advantage of specifying operations of UCAADS and modeling user(s) and user-device interactions in terms of workflows is that such specifications and models are executable. The developer of the new or modified device can thus specify the device and model user(s) and device-user interactions as soon as the requirements of the device have been captured and a preliminary design is available. By executing the models, the developer can assess the design, usability and performance of the device at each stage of the development process. Indeed, we have used UCAADS models in simulation experiments to assess the flexibility of new devices and to expose their design and implementation flaws and unsafe user actions, in addition to assessing the responsiveness of the devices [4]. The UCAADS Simulation Environment (USE) [2, 3] is for this purpose.

The reminder of the paper is organized as follows. Section 2 describes the elements of the UCAADS model and illustrates their use. Section 3 provides an overview of the USE and describes how the UCAADS model and USE can be used to support the development of UCAADS. Section 4 presents an overview of EMWF. Section 5 summarizes the paper and presents our future plans.

II. Elements of UCAADS Model

As stated earlier, the UCAADS model of a device (or system) and its user(s) captures the behavior of the device and actions (operations) of its user(s) in terms of device and user workflows. The model also incorporates elements of GOMS and MHP [23, 24] as building blocks of user workflows and thus enables the developer to model user actions more precisely. Specifically, we define human operators (i.e., perceptual, cognitive, and motor operators) in terms of workflows so that they can be executed during simulation for the purpose of estimating execution times of user actions for different users and for each individual user under different conditions.

A. *Workflow Elements*

There are two types of workflows, sequential workflow and state machine workflow. A sequential workflow consists of a fixed sequence of activities. The execution path may branch, or loop, etc. but has a workflow-wide starting point and ending point. 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 state transitions, are triggered by external events. We use both types of workflows in behavior specifications of devices and models of users and their actions.

As an illustrative example, Figure 1 shows the load-pantry part of the behavior specification of a smart pantry that identifies objects stored in it by their bar-code ids [8]. The user workflow in left side of the figure is sequential. The activities in it model user actions in the load-pantry process during which the user puts objects into the pantry. The workflow in the right side of the figure is also sequential. It is the LoadPantry device workflow that specifies device behavior and service in response to the user actions.

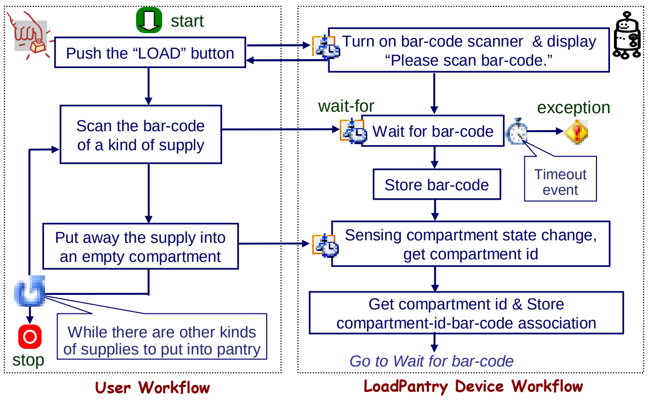

start
Push the "LOAD" button
Turn on bar-code scanner & display "Please scan bar-code."
wait-for
Scan the bar-code of a kind of supply
Wait for bar-code
exception
Timeout event
Store bar-code
Put away the supply into an empty compartment
Sensing compartment state change, get compartment id
Get compartment id & Store compartment-id-bar-code association
While there are other kinds of supplies to put into pantry
stop
Go to Wait for bar-code
User Workflow
LoadPantry Device Workflow

Figure 1. LoadPantry operation, an example

A user starts a load-pantry process 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 id 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 pantry stores the association between the bar-code id 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 ids.

B. GOMS Elements

Oftentimes, the developer also wants to know the lengths of time required to complete important operations. Take a multiple-user medication station [11] as an example. Its users are nurses. The station can operate semi-automatically and collaborate with the users and their mobile tools to ensure that right medications are retrieved by the right users for the right patients. It is important to know the maximum number of nurses the station should allow to retrieve medications stored in it concurrently. The response time for each user depends not only on the number of users and their interactions with the system during the retrieval process, but also on the execution times of user actions. By incorporating well-known GOMS models of human behavior with the workflow models of user actions, we can more precisely model the users and estimate the execution times of their actions.

GOMS has been widely used for years in studies on human-computer interactions. USE supports the 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.

CPM-GOMS variant supports concurrent executions of human operators. Among the variants of GOMS, it is most compatible with workflows and more natural for modeling many UCAADS user actions. To incorporate CPM-GOMS model with workflows, we implemented basic and frequently used human operators (e.g., perceptual, cognitive and motor operators) as human-operator activities: The parameterized functions executed by the activities implement the rules (e.g. Fitt's law for moving a hand) and parameters (e.g., start and end positions of the mouse cursor) that define the operators and the calculation of execution or learning times of users. A developer can use these human-operator activities as components of composite activities and sub-workflows that represent simple user actions (e.g., push a button).

To illustrate, Figure 2 shows a fragment of CPM-GOMS model of a user pressing the LOAD button on the pantry. Except for precedence constraints indicated by edges in the graph, operators can execute in parallel. Figure 3 shows a workflow model fragment that "implements" the CPM-GOMS fragment. For sake of simplicity, cognition and visual perception operators are implemented by a single sub-workflow, called cognition & vision sub-workflow. The actions of eyes and hand 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 the user. The sub-workflows synchronize via the wait event built-in activity. In this way, we capture in workflows the precedence constraints in the CPM-GOMS fragment.

C. Local service for device-user interaction

The third element of UCAADS model is the USE interface service. The service is provided by the simulation environment for communication between user workflows and device workflows in a behavior specification. It is built on the custom data exchange services supported by WF. Figure 4 shows the essence of the service.

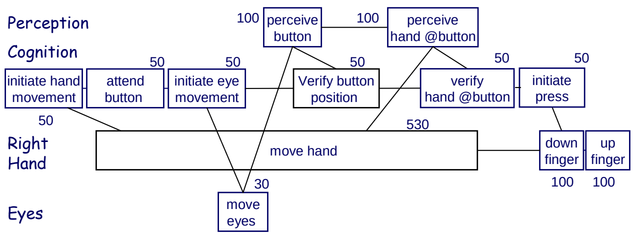


Figure 2. CPM-GOMS model of "Push LOAD button"

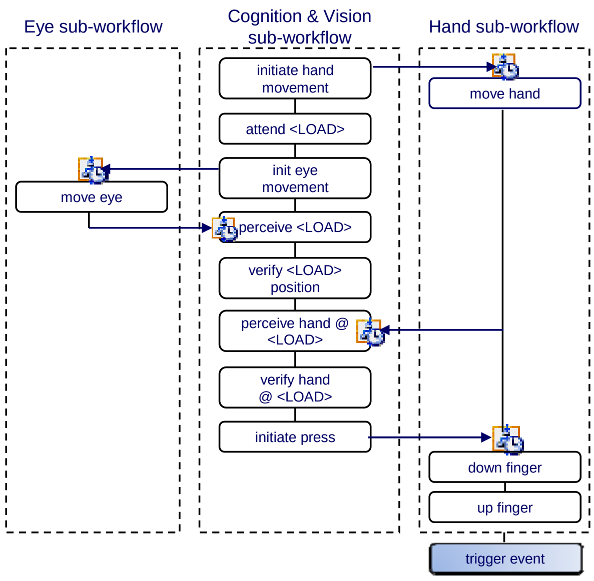


Figure 3. "Push LOAD button" CPM-GOMS template in workflow

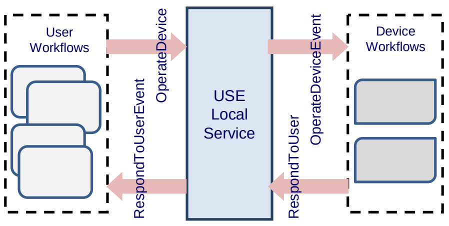


Figure 4. USE local service

Specifically, USE local service enables workflow instances to invoke each other via events. After a workflow instance 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. A user workflow instance calls OperateDevice( ) method to raise an OperateDeviceEvent event. The event argument carries information on a user action (e.g., click, push, open, etc.), the target of the action (e.g., scanner, LOAD button, drawer #8, etc.), and other information (e.g., bar-code id, medication-id) associated with the action. The local service delivers the event to the device workflow instance specified by the deviceIdentifer argument. In response, the device workflow instance calls RespondToUser( ) method to raise a RespondToUserEvent event and have the local service delivered the event to a user workflow. The event carries the output type of the response and the message of the response

III. USE AND UCAADS DEVELOPMENT PROCESS

Figure 5 shows the structure of USE. Again, a developer specifies the behavior of the device or system being evaluated by device workflow(s) and user actions by user workflow(s). The run-time environment takes them as inputs of simulation experiments.

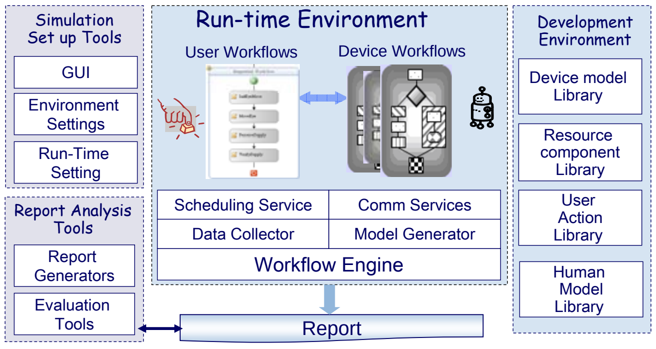


Figure 5. USE architecture

A. Model Libraries

As the figure shows, USE has four extensible libraries. The device model library contains reusable activities and workflows (e.g., load pantry workflow in Figure 1) for constructing device workflows. The resource component library holds dynamically linked library (DLL) functions, executable and other types of resources required by activities in device workflows. As examples, many UCAADS (e.g., medication dispenser [5], iNuC (intelligent nursing cart) [11], and MUMS (Multi-User Medication Station) [12]) have drawers, a sensor on each drawer for detecting whether the drawer is open or closed, and, an interlock mechanism which, when commanded, opens a locked drawer specified by a bar-code id. Drawer, sensor and actuator hardware and drivers are resource components required by the device workflows that handle the event raised when a drawer changes state and the user's action of scanning a bar-code id in order to open a locked drawer. Before the actual hardware and drivers are available, the resource component library contains simulators/emulators of these components.

User action library holds activities and workflows (e.g., human-operator activities and the push LOAD button CPM-GOMS workflow) from which developers can construct user workflows. Most of the resources required by user activities and workflows are DLL functions.

The human model library contains human models. These 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.

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 6 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.

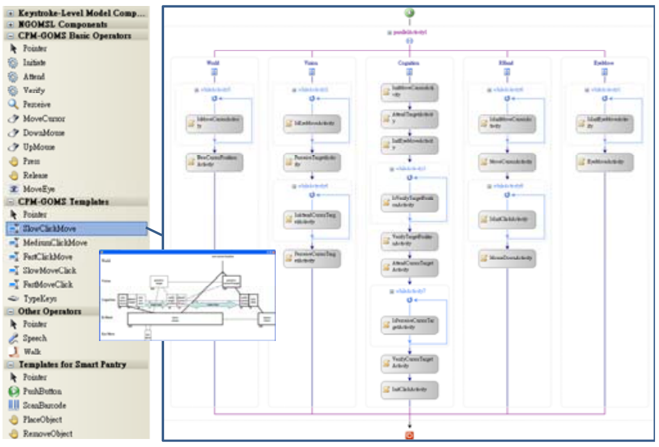
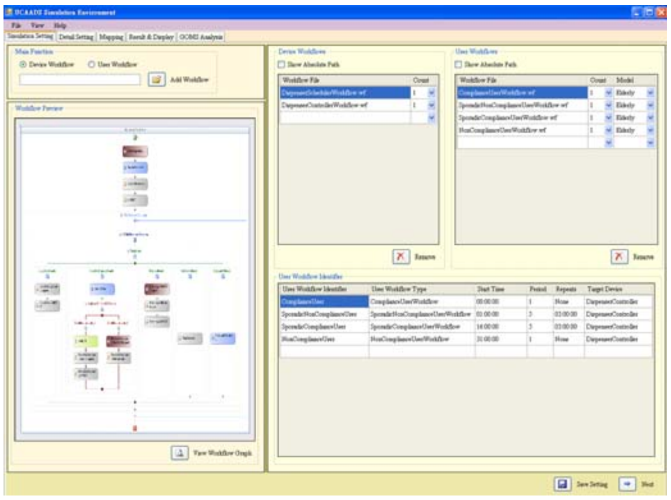


Figure 6. Human action models in toolbox

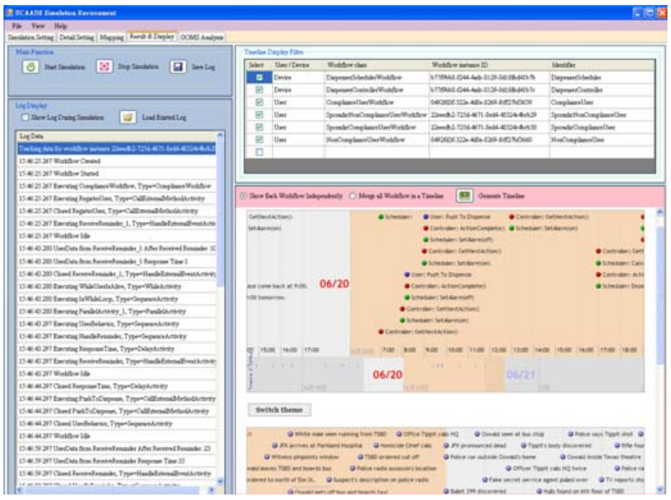
B. Evaluation Tools and Data Collector

It is well known that human errors may in fact be design flaws and errors of devices. When a user operates a device, he/she expects that the device to enter some state or delivers a certain service. Discrepancy in the expectation and view of the user and the actual state and behavior of the device is known to have caused serious errors, accidents, and harm. To help the developer identify and eliminate user-device interaction that can cause this kind of problem, USE provides a 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 activity completes. 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.

As a part of the run-time environment, 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 and device workflows. The data are stored in a database for off-line analysis and display. Figure 7 shows the part of USE GUI for setting up a simulation run and displaying the 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 generated from the recorded simulation data by an open source timeline widget [26] that is embedded in USE.



(a)



(b)

Figure 7. USE GUI

C. *Model-Based Development*

From [1, 4] and its name, one can see that our initial intention was to use USE solely for simulation and evaluation purposes. For workflow-based devices and systems (e.g., iNuC version 1.5 and MeMDAD component systems [11, 12]) that run on WF, the current version of USE is also an integrated environment capable of supporting model-based design, implementation and quality assurance of the devices/systems.

To be concrete without loss of generality, we suppose that the goal of the first phase of the development process is to capture the requirements and produce a design and behavior specification of the new device. For many UCAADS (e.g., iNuC and MeMDAS), an effective way to accomplish this goal is to use a mockup. The mockup is a virtual device that behaves to the user as a real device. (Take the iNuC mockup as an example. It gives the user the look and feel of a real iNuC except for the absence of physical medication drawers with interlock mechanism and real patient data and nurse personal information.) We have found mockups effective as tools for acquisition of accurate information on what the users want the new device to do and how they want the device to interact with them, etc. Moreover, through the evaluation process involving actual users using a mockup, most if not all the inconsistencies between views of the user and the system can be identified and eliminated. USE and UCAADS model can help to reduce the effort in producing a high-quality mockup of a new device.

Specifically, within the framework of UCAADS model, a mockup of a device is composed of state machine and sequential workflows that depict the device behaviors, including activities for handling events trigged by user actions. They form the behavior specification of the device. Many UCAADS (e.g., iNuC and MeMDAS) have complex GUI. At least, a preliminary version of the GUI must be included in the mockup to be evaluated by actual users. An advantage of building the mockup from workflow models is that it can be tested and evaluated before the GUI is ready using user workflows to model actions of all kinds of users. Such tests enable us to explore all the possible user scenarios and make sure that the device handles these scenarios in correct manner. In this step, device workflows and user workflows interact through USE local service described above.

In the implementation phase, more and more physical devices and programs required by workflows in the behavior specification of the device become available. The behavior specification of the device becomes an implementation of the device when all the device workflows switch from using virtual resources to actual resources. While the GUI is still being refined and remains unavailable for testing purpose, user workflows can be used to trigger events that simulate events from the GUI raised in response to user actions that operate the interface. Similarly, user workflows can be used to trigger events on USE local service to simulate user actions that are not GUI related (e.g., open a drawer and scan a bar-code).

In the final phase of the development process, the new device is almost completed. The test team can test the GUI, device workflows and the local service supporting the GUI-workflow communication as a whole. In particular, once the GUI is completed, user workflows can be used to simulate GUI operations by operating the GUI directly.

IV. EMBEDDED WORKFLOW FRAMEWORK

We chose to work with Microsoft .NET WF because the tools and services available within WF can significantly shorten our prototyping time. Unfortunately, the workflow management system has many shortcomings for UCAADS. Similar to workflow management systems running in J2EE environment (e.g., [20]), WF requires large amounts of system resources. Many capabilities and features provided by .NET and WF are essential for general workflow applications but not for workflow-based UCAADS. At the same time, it does not provide the developer with sufficient control over scheduling and resource management needed to ensure good real-time performance. These shortcomings motivated us to develop the embedded workflow management system EMWF [3] with the goal of replacing WF as the middleware of choice for UCAADS. All versions of EMWF are written in C in order to keep the memory consumption and runtime overhead introduced by the engine small. Both EMWF versions 1.0 and 2.0 run on Microsoft Windows Embedded Standard.

A. *EMWF 1.0*

Specifically, EMWF 1.0 is for relatively simple embedded devices and system components that run on a processor. The embedded workflow definition language supported by EMWF 1.0 is called SISARL-XPDL. It is composed of a subset of the WfMC standard XPDL 2.0 [17] and an extension. The extension includes built-ins activities needed for robot behavior coordination, for message passing in distributed environments, and by multi-mode embedded applications. It also includes Period and ExtendedAttributes. These elements enable the developer to specify which workflows are real-time, what their execution rates are, and what custom scheduling policy is to be used if the workflows are not to be scheduled by default on rate-monotonic basis. The SISARL-XPDL parser first translates extension elements into standard XPDL 2.0 elements and then compiles XPDL 2.0 definitions into executable workflow scripts. Workflow scripts are stored in .wfs files.

Figure 8 shows the general structure of workflow-based devices running on EMWF 1.0 from the perspective of the workflow engines. Specially, it shows the major components of EMWF 1.0, including engine manager, workflow manager and workflow processor.

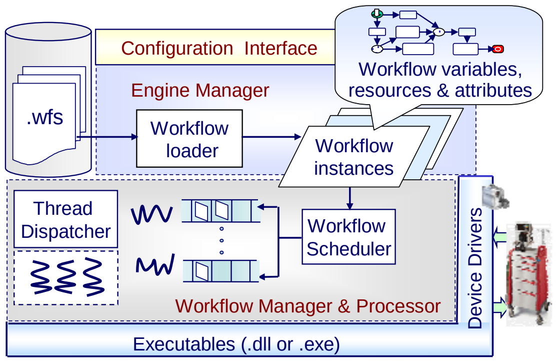


Figure 8. Major components of EMWF

The engine manager is responsible for handling user requests and managing their accesses to workflow-related definitions and optional contextual information. The developer can tune the engine via the configuration interface. Configuration parameters include the maximum numbers of threads and priority levels and the finest resolution of timers. During initialization, the engine manager initializes and configures the engine. It then loads the .wfs files needed by all the applications for all operation modes into memory.

EMWF 1.0 offers two different multi-threaded workflow processors. They are the WLA (workflow-level assignment) engine and the ALA (activity-level assignment) engine. Figure 8 shows the structure of the WLA engine: Each thread is dedicated to execute a workflow instance. The workflow manager attached a thread to each workflow instance and schedules the thread to execute all activities in it at the priority of the instance. An ALA engine uses worker threads to execute activities as work items: The workflow manager maintains a FIFO queue per priority for queuing work items and assigns at least a thread to each queue. The thread (or threads) serving a queue executes at the priority of the queue.

B. *EMWF 2.0*

To make EMWF better suited for distributed applications (e.g., MeMDAS) and real-time applications (e.g., delivery robots), EMWF 2.0 extended EMWF 1.0 in three ways: The first is a general messaging passing mechanism [3] needed to support push data and pull data built-in activities.

The second is a two-level scheduling mechanism for end-to-end scheduling in distributed environments. Figure 9 shows the structure of a ALA engine containing the two-level scheduler. We note that the scheduler should be able to support well-known end-to-end real-time scheduling schemes used in statically configured systems. The workflow manager calls the high-level end-to-end scheduler when a new workflow instance becomes ready for execution. Currently, the only task performed by the scheduler is to distribute the available end-to-end slack time of each workflow (or chain of workflows) to activities or chains of activities according to specified algorithm(s). Both low-level schedulers (i.e., activity scheduler for CPU scheduling and message scheduler for network traffic) support fixed priority scheduling.

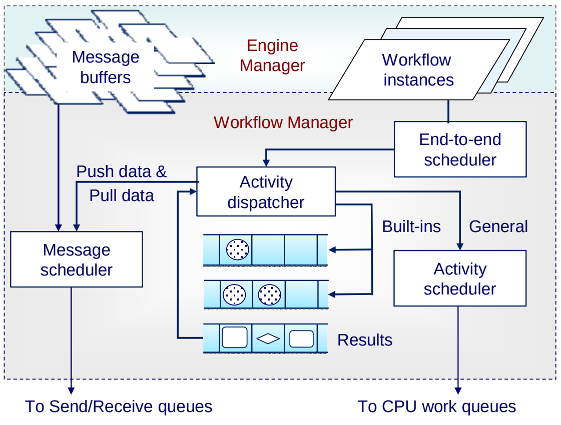


Figure 9. Structure of two-level scheduler

The third extension in EMWF 2.0 is a service interface designed to support workflow-to-workflow interactions and their interactions with their host application. Figure 10 gives an overview to illustrate ways workflow instances and non-workflow components communication and interact, making use of the service interface. Rectangular shapes represent service interfaces, host application and non-workflow components. Polygons represent workflow instances of the same type or different type. Bold arrows represent callbacks, simple arrows with solid lines represent interface function calls, and arrows with dotted lines represent raises and deliveries of events.

EMWF service interfaces resemble closely to local services of WF. The kind of assistance EMWF 2.0 will provide to service interfaces is similar to what WF does for local services: In particular, each service interface is identified by its type. After the developer has declared a service interface type, implemented a service interface of the type and have the application created and registered the service interface with the workflow management runtime during initialization time, workflow instances in the application can query the workflow manager for functions provided by the service and make use of the functions for communication and invocation.

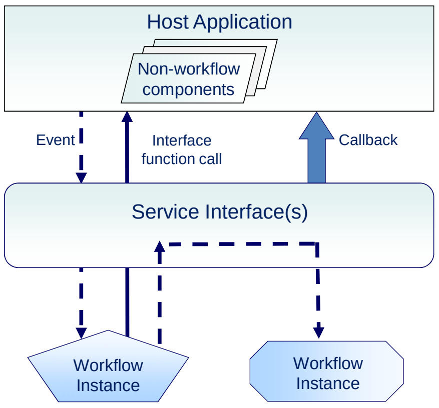


Figure 10. Interaction via serive interfaces

Specifically, workflow instances invoke each other and deliver results to each other by raising events. In essence, the system of service interfaces serves as a router, routing each event raised by a workflow instance to one or more workflow instances as specified by the parameters of the raise-event interface function. In a distributed system, the workflow manager tracks the locations of all workflow instances and thus relieves the developer from the burden of this work.

In addition to events, workflow instances can communicate with the host application and other non-workflow components via callbacks. We note that when a caller raises an event to invoke a workflow instance, the event handler is executed by the thread dispatched to execute the instance. When the workflow instance responds to a caller via a callback function, the function is also executed by this thread. It is important to follow the principle of this pattern when the caller must be responsive.

V. Summary and Future Plans

We described here how the combination of workflow model and GOMS model provides a flexible way to define device operations and the user actions for the purpose of evaluating UCAADS and their user-device interactions. The UCAADS simulation environment provides libraries of reusable model components for the construction of models and behavior specifications of new and modified devices and supports prioritized execution of workflows in the models.

The simulation results can help us in two aspects throughout the UCAADS development process. The first is refinements of the device design. As an example, in simulation experiments on the medication dispenser described in [5, 6], we examine the protocol between the controller and the scheduler of the dispenser and make sure that the dispenser handle all non-compliance events correctly. In experiments on the smart pantry [8], we found that the smart pantry may fail and order wrong supplies 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 [12], 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.

There are many works remain to be done to improved USE as a modeling and evaluation tool. As examples, we want to design a script language and exploit current GOMS automation tools to automatically generate user workflows from the human action library. We want to provide USE with the capability of automatically generating GOMS models by simply demonstrating tasks via a graphic user interface that are implemented in Windows Form and Windows Presentation Foundation in the future. We plan to improve the evaluation and analysis tools to reduce developers' efforts in finding complicated error scenarios.

We were drawn to Microsoft .NET WF when choosing a workflow management system for USE not because of the workflow engine itself but for all the support tools it provides: They indeed significantly reduce the efforts and times we took to develop iNuC and MeMDAS, in addition to USE. In this respect, EMWF 2.0 falls short as a development environment.

We have not yet explored the full potential of USE as a model-based development environment of workflow-based devices and systems. Rather, most part of our effort in this direction has been directed to improving EMWF in an attempt to make it the platform of choice for small embedded devices as well as sizable systems such as MeMDAS and various forms of service robots. Our ultimate goal is to replace WF by EMWF 2.0. Currently, the real-time scheduling mechanism for end-to-end scheduling is operational but the interface service for workflow-to-workflow interaction is not yet implemented.

Acknowledgment

The authors wish to thank T. S. Chou for his contribution on design and development of EMWF 1.0, and C. H. Chen and C. W. Yu for their help in workflow implementation of CPM-GOMS models and USE GUI.

References

[1] T.Y. Chen, P. H. Tsai, T. S. Chou, C. S. Shih, T. W. Kuo, and J. W. S. Liu, "Component Model and Architecture of Smart Devices for the Elderly," *Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture*, pp. 51 – 60, February 2008.

[2] T. Y. Chen, C. H. Chen, C. S. Shih, J. W. S. Liu, "A Simulation Environment for the Development of Smart Devices for the Elderly," *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, October 2008.

[3] T. S. Chou, H. Y. Huang, Y. C. Wang, W. S. Chen, C. S. Shih, and J. W. S. Liu, "EMWF: A Middleware for Flexible Automation and Assistive Devices," *Proceedings of the 8th IFIP Workshop on Software*

Technologies for Future Embedded and Ubiquitous Systems, October 2010.

[4] T. Y. Chen, P. H. Tsai, C. H. Chen, C. W. Yu, C.S. Shih and J. W. S. Liu, "A Model and Simulation Environment for Symbiotic Automation and Assistive Devices," Technical Report No. TR-IIS-10-008, Institute of Information Science, Academia Sinica, October 2010.

[5] P. H. Tsai, 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," *Proceedings of IEEE Healthcom*, July 2010.

[6] P. H. Tsai, C. Y. Yu, C. S. Shih and J. W. S. Liu, "Smart Medication Dispensers; Design, Architecture and Implementation," *IEEE Systems Journal*, March 2011.

[7] T. S. Chou and J. W. S. Liu, "Design and Implementation of RFID-Based Object Locator," presented at the 2007 IEEE International Conference on RFID, 2007.

[8] C. F. Hsu, et al., "Smart Pantries for Homes," *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, 2006.

[9] Y. Kaneshige, et al., "Development of New Mobility Assistive Robot for Elderly People with Body Functional Control," *Proceedings of the 1st IEEE RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, Vols 1-3, 2006, pp. 460-465.

[10] J. Forlizzi and C. DiSalvo, "Service Robots in Domestic Environment: A Study of Roomba Vacuum in the Home," *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, 2006.

[11] P. H. Tsai, et al., "iNuC: An Intelligent Mobile Nursing Cart," *Proceedings of the 2nd International Conference on Biomedical Engineering and Informatics*, Vols 1-4, 2009.

[12] J. W. S. Liu, et al., "MeMDAS: Medication Management, Dispensing and Administration Systems," presented at mHealth Workfshop, IEEE HealthCom2010, June 2010.

[13] SpeciMinder - hospital delivery robot, http://speciminder.com/

[14] TUG – automated robotic delivery system, http://www.aethon.com/

[15] B. Bukovics, Pro WF: Windows Workflow Foundation in .Net 4.0, Apress 2009.

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

[17] XPDL (XML Process Definition Language) Document, http://www.wfmc.org/standards/docs/TC-1025_xpdl.2.2005-10-03.pdf, October 2005

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

[19] Open Source Java XPDL editor,

[20] Enhydra Shark, http://forge.objectweb.org/projects/shark

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

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

[23] B. E. John and D. E. Kieras, "The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast," *ACM Trans. Computer-Human Interaction*, vol. 3, pp. 320-351, 1996.

[24] S. K. Card, et al., *The Psychology of Human-Computer Interaction:* Lawrence Erlbaum Associates, 1983.

[25] T. Y. Chen, et al., "Model-based design, implementation and evaluation of UCAADS," Technical Report No. TR-IIS-11-002, Institute of Information Science, Academia Sinica, in preparation.

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