



中央研究院
資訊科學研究所

Institute of Information Science, Academia Sinica • Taipei, Taiwan, ROC

TR-IIS-09-009

EMWF: A Middleware for Flexible Automation and Assistive Devices

J.W. S. Liu, C. S. Shin, T. S. Chou, Y.C. Wang, H. Y. Huang, W. S. Chen,
K. C. Chuang, W. C. Wang and T. Y. Chen



December 1, 2009 || Technical Report No. TR-IIS-09-009

<http://www.iis.sinica.edu.tw/page/library/LIB/TechReport/tr2009/tr09.html>

Institute of Information Science, Academia Sinica
Technical Report TR-IIS-09-009

EMWF: A Middleware for Flexible Automation and Assistive Devices

J. W. S. Liu, C. S. Shih, T. S. Chou, Y. C. Wang, H. Y. Huang, W. S. Chen,
K. C. Chuang, W. C. Wang and T. Y. Chen

ABSTRACT

EMWF (embedded workflow framework) is an open source middleware for flexible (i.e., configurable, customizable and adaptable) automation and assistive devices and systems, referred to collectively as SISARL (Sensor Information Systems for Active Retirees and Assisted Living). Examples include smart medication dispensers, autonomous appliances, service robots and robotic helpers for personal and home use, as well as automation tools for use in hospitals and long-term care facilities. EMWF 1.0 provides a light-weight workflow manager and engines on Windows CE, Windows XP Embedded, and Linux. It is for small embedded automation devices. EMWF 2.0 will include basic message passing mechanism, real-time scheduling capability and workflow communication facility. This paper first gives an overview of EMWF 1.0 and then describes these extensions.

Copyright @ November 2009

J. W. S. Liu and Y. C. Wang are affiliated with Institute of Information Science and Center for Information Technology Innovation, Academia Sinica, Taiwan.

C. S. Shih, W. S. Chen and K. C. Chuang are affiliated with Computer Science and Information Engineering Department, and W. C. Wang is affiliated with Electrical Engineering Department, National Taiwan University, Taiwan.

T. S. Chou, H. Y. Huang and T. Y. Chen are affiliated with Department of Computer Science, National Tsing Hua University, Taiwan.

TABLE OF CONTENTS

<i>ABSTRACT</i>	<i>1</i>
<i>TABLE OF CONTENTS</i>	<i>2</i>
<i>1 INTRODUCTION</i>	<i>3</i>
<i>2 MOTIVATIONS AND RATIONALES</i>	<i>4</i>
(A) Simple Workflow-Based Devices	<i>5</i>
(B) Messaging and Real-Time Capabilities	<i>6</i>
(C) Workflow Communication	<i>8</i>
<i>3 EMWF 1.0 OVERVIEW</i>	<i>10</i>
(A) SISARL-XPDL	<i>10</i>
(B) Engine Manager	<i>11</i>
(C) Workflow Manager and Processor	<i>12</i>
(D) Relative Merits	<i>13</i>
<i>4 EXTENSIONS IN EMWF 2.0</i>	<i>15</i>
(A) Messaging Mechanism	<i>15</i>
(B) End-to-end Scheduling	<i>17</i>
(C) Service Interfaces	<i>19</i>
<i>5 SUMMARY</i>	<i>21</i>
<i>6 ACKNOWLEDGMENT</i>	<i>22</i>
<i>7 REFERENCES</i>	<i>23</i>

1 INTRODUCTION

A major thrust of our recent research has been directed towards advancing the technologies for designing and building high-quality personal and home automation and assistive devices and systems at low cost. We refer to these devices (and systems) collectively as SISARL (Sensor Information Systems for Active Retirees and Assisted Living). Examples include smart medication dispensers, autonomous appliances, service robots and robotic helpers (e.g., [1-11]) designed to improve the quality of life and self-reliance of elderly, chronically ill or functionally limited individuals. SISARL also include automation and point-of-care tools (e.g., [12- 14]) for use in hospitals and other care-providing institutions for enhancing the quality and reducing the costs of medical and health care.

Despite vast differences in their purposes and functions, SISARL have many common requirements. First and foremost is *flexibility*; by that, we mean configurability, customizability and adaptability. A flexible device can be easily configured to work with a variety of sensors and devices, rely on different support infrastructures and operate in different environments. It should be easily customizable according to its user's preferences. The device should be able to adapt to serve the user well over time as the need of the user changes.

We have adopted the workflow approach [15] as a means to achieve flexibility. Basic building blocks of a workflow-based device are *activities*: They are elementary steps of work done by the device. Some activities are executable code running on one or more CPU or microcontroller. Some activities are done by sensor devices, special purpose hardware and mechanical components. A semi-automatic device also has activities that are done by the user(s). Activities are composed into module-level components called *workflows*. The order and conditions under which activities in a workflow are executed, the resources (also called *participants*) needed for their execution, and interactions among activities are defined by the developer of the workflow. The definition can be in terms of some programming language (e.g., C# in Windows Workflow Foundation [16]), a process definition language (e.g., XPDL, WfMC standard XML Process Definition Language [17, 18]), or an execution language (e.g., BPEL, Business Process Execution Language [19]). Workflows can also be defined graphically [20]: In a workflow graph, nodes represent activities or states of the device, and directed edges represent transitions between activities or state transitions caused by executions of some activities..

To design and implement a workflow-based application, the developer only needs to define the workflows in the application and provide the resources (including executable library

functions, hardware devices, etc.) required by carried out the activities in them. Execution, sequencing and synchronization of activities and workflows are done by a middleware component called *workflow engine* (or engine for short). In essence, the workflow engine integrates workflow components dynamically at runtime as specified by the developer. EMWF (Embedded Workflow Framework) [21] is such a middleware. Like other workflow management system, EMWF also provides a *workflow manager* for scheduling activities and workflows, allocating resources to them and facilitating communication among them.

Because of the relative ease with which workflow-based applications can be designed, implemented and configured, the workflow approach has been widely used in enterprise computing systems for automation of business processes. Today, there are standard process definition languages and execution languages, as well as matured engines and tools for defining, building and executing workflow applications (e.g., [16-20, 22-25]) on enterprise computers and mobile devices. They enable business application developers and domain experts with limited information technology expertise to tailor complex business processes to individual enterprises and across enterprises. EMWF aims to enable embedded devices be built on workflow paradigm. Case studies on modeling and design of workflow-based SISARL and their development and evaluation [21, 26, 27] have demonstrated to a great extent that componentization and flexibility come naturally with workflow-based design for embedded applications as well.

Following this introduction, Section 2 provides illustrative examples to further elaborate workflow-based design and discusses rationales behind EMWF. Section 3 provides an overview of EMWF version 1.0 [21]. Section 4 describes extensions designed to provide the workflow management system with communication and real-time capabilities and additional requirements of EMWF 2.0, the next version of EMWF. Section 5 summarizes the paper.

2 MOTIVATIONS AND RATIONALES

Specifically, EMWF provides a workflow manager and engines on Microsoft Window CE , Windows XP Embedded and Linux. The engines are written in C in order to keep the memory footprint and runtime overhead introduced by the engine small. We focus here on the relatively mature Windows versions. Hereafter, by EMWF, we mean these versions except for where it is stated otherwise. Descriptions of a Linux version can be found in [21, 28].

EMWF engines are designed for embedded applications containing workflows that may run at high rates and interact closely with hardware devices. For this reason, existing workflow management systems for web-based business applications are not suitable for them. Typical

SISARL devices are not as severely power and size constrained as cell phones and PDA's. Consequently, energy consumption and memory footprint requirements for EMWF engines and workflow applications are not as stringent as the requirements of engines and applications for mobile workflow management (e.g., [23-25]).

(A) Simple Workflow-Based Devices

Specifically, EMWF 1.0 is suited for relatively simple devices and system components that run on a processor. An example is an automatic vacuum cleaner. Its workflow-based structure is shown in Figure 1 for illustrative purpose. Most parts of the devices are built from workflows. We omit drivers and components that are hardwired. The rectangular boxes represent activities. The activities in the middle dotted box are executed by the workflow engine on a CPU. We call them *software activities*. Embedded devices also have *external activities*. In this example, they are in dotted boxes labeled environment interaction and robot components. External activities are carried out by sensor devices, microcontroller and mechanical parts of the device.

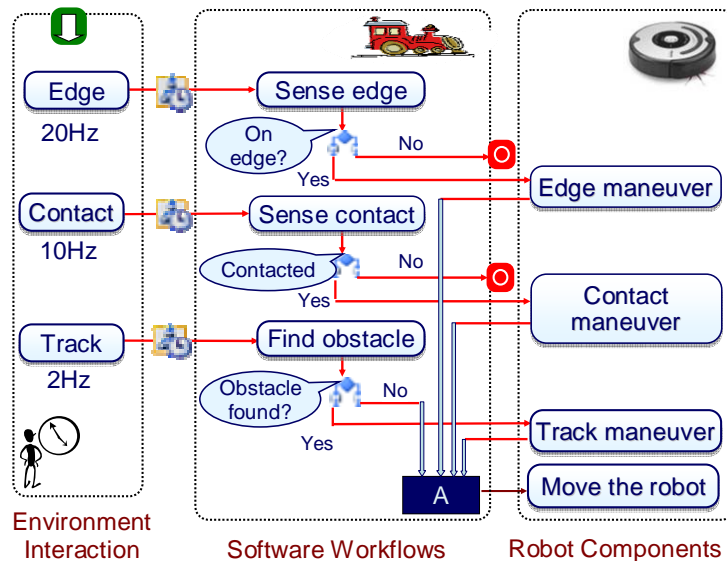






















Figure 1 Example of a workflow-based device

Table 1 lists activities that start and stop workflows, and alter the timing and flow paths during their executions. These activities are said to be *built-ins* because they are provided by EMWF. The symbols used to represent some of them are from Microsoft Windows Workflow foundation (WF) GUI editor [16]. Top five rows lists examples of generic built-ins, including start, stop, if else, and wait-for built-ins shown in Figure 1. All applications need some or all generic built-ins. The bottom row of Table 1 lists built-ins specifically for robotic behavior coordination (BC), including arbiter, the built-in that performs fixed-priority arbitration among

edge, contact and track maneuvers in the device in Figure 1. Arbiter is implemented in EMWF 1.0 with generic built-ins as described in [21]. Table 1 also lists built-in activities to be provided by EMWF 2.0, including superposition and voter activities for behavior coordination in robotic applications and push data, pull data and mode change activities for all embedded applications.

Table 1 Built-in activities

Generic built-ins	 Route	 If else (2-way XOR split)	
	 Split	 Merge	
	 Throw	 Exception	
	 Invoke workflow	 Execute workflow	
	 Delay /timeout	 Set events / timers	
	 Wait for events / timers / workflow triggers		
	 Start	 Stop	 While
Built-ins for BC	 Superposition	 Arbiter	 Voter
	 Mode change	 Push data	 Pull data

Before moving on, we note that one can easily build devices with different functionalities from components by modifying the graphs defining their workflow processes and/or using different participants for activities. Indeed, using the same reconfigurable workflows, we can change the workflow-based vacuum cleaner shown in Figure 1 into a navigation and propulsion component of an intelligent nursing cart by simply changing the workflow graph. By replacing the “back and random move” contact maneuver activity used in the vacuum cleaner with a “move back and hit” activity, we can change the device and make it behave like a toy sumo.

(B) Messaging and Real-Time Capabilities

Many SISARL applications rely on multiple computers, wire and wireless networks, local and remote sensors and control devices, and mobile and fixed user interfaces. Examples include many mobile robots and automation devices. To illustrate, Figure 2 shows a block diagram from [29] containing some of the tasks in Pygmalion, an experimental robot capable of performing delivery services. If the robot were workflow based, these tasks would be implemented by workflows. According to [29], the bumpers drivers and speed controller run at 1 KHz, and the position controller runs at 50 Hz. Obstacle avoidance makes use of the vision system and laser range finder. The former processes images captured by a camera to extract vertical lines.

Whether a vertical line is an obstacle is determined with the help of the range information captured by the laser range finder. Such a robot may also have a speech recognition system so that it can capture and interpret voice commands and a speaker location system that pinpoints where the speaker is by processing the delay patterns of direct and reflected sound signals. Oftentimes, navigation and obstacle avoidance tasks run on a processor while vision system and speech related tasks run on a separate processor or processors, and speed and position control tasks run on yet another processor.

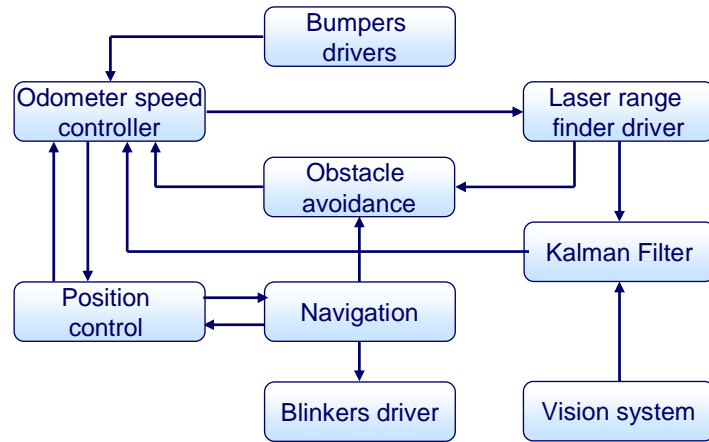


Figure 2 Tasks of a mobile service robot [29]

To support applications exemplified by Pygmalion, we extended EMWF 1.0 with a low-level message passing mechanism that implements push data and pull data built-ins in Table 1. It provides the essential low-level support for an end-to-end distributed workflow management system over all resources to facilitate the collection of data from sensors, delivery of commands to controllers, and monitoring and processing feedbacks from physical processes and so on. Section 4 will describe the mechanism.

Many tasks shown in Figure 2 have real-time requirements: Obstacle avoidance and position control are examples. They must complete on a timely basis for the robot to move smoothly at a required speed without bumping into obstacles. For such devices, a defect of existing workflow management systems, include EMWF 1.0, as well as existing middleware for robotic applications [30], is the lack of adequate real-time support. This is the reason for extending EMWF 1.0 with an end-to-end scheduler at the higher level and a message scheduler at the lower-level. Together with prioritized queues provided by the workflow manager in EMWF 1.0 for priority-driven CPU scheduling, the two-level scheduler enables the extended EMWF 1.0 to support many well known end-to-end scheduling strategies (e.g., [31, 32]).

(C) Workflow Communication

EMWF 1.0 supports event-driven, sequential workflows, but not state machine workflows. More seriously, it lacks easy-to-use facilities for invocation of workflows by workflows and data exchanges between workflows. These limitations prevent us from using it for complex SISARL devices such as iNuC (intelligent nursing cart) [14] shown in Figure 3.

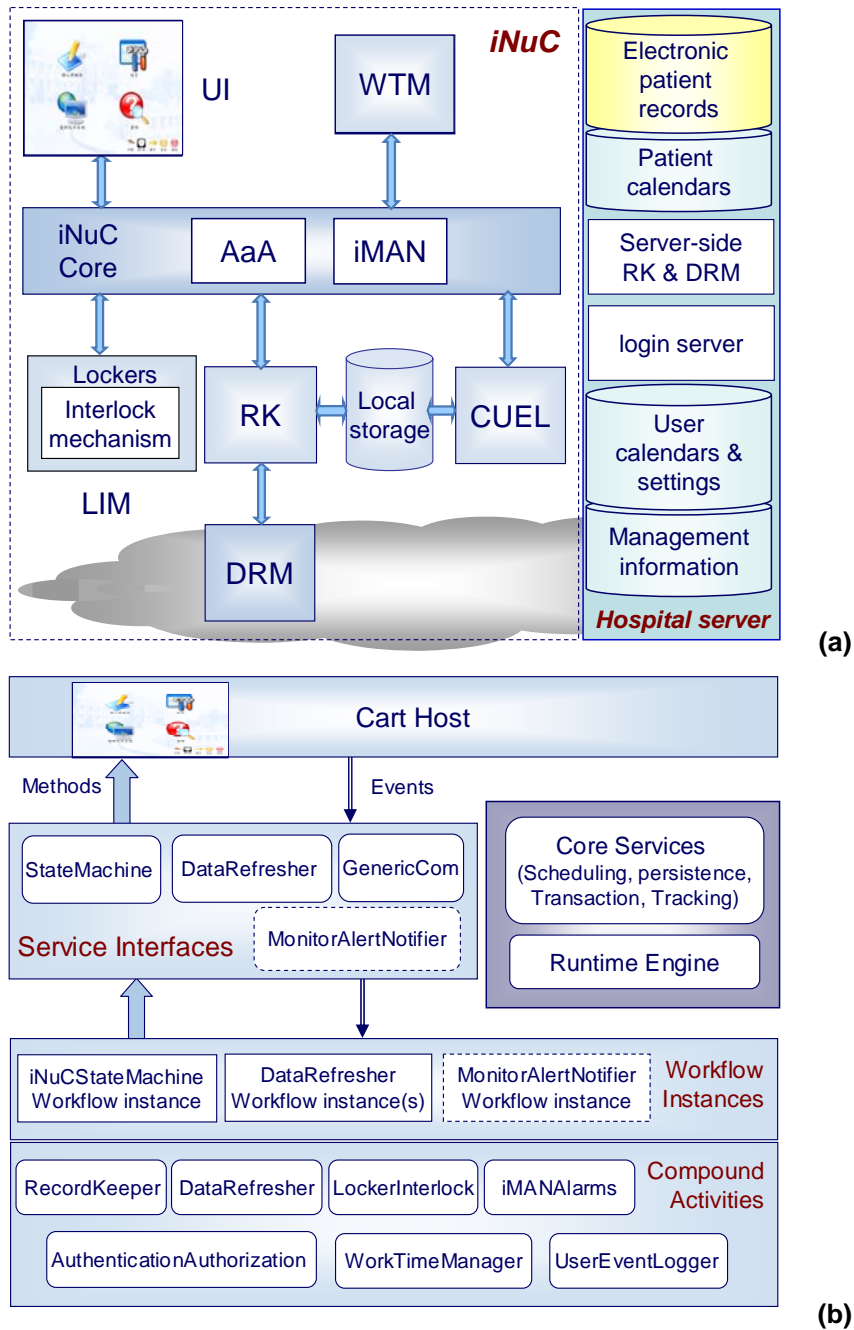


Figure 3 Alternative architectures of iNuC

Specifically, iNuC is a mobile medication administration and record keeping tool for nursing staffs. Such a cart carries daily supply of medications for each patient cared by the cart user. It is designed to be a self-contained unit: During network and hospital-wide server outage, the cart reliably maintains all records and synchronizes with the server when the server becomes available again. The major component tools and system modules include the user interface (UI), authentication and authorization module (AaA), work-time manager (WTM), intelligent monitor, alert and notification (iMAN), cart locker interlock mechanism (LIM), record keeper (RK), data refresh mechanism (DRM) and cart user event log (CUEL). Figure 3 (a) shows these components and the hardwired structure of the current version.

A hospital typically needs not only full-service carts like iNuC, but also basic mobile carts (BaMC). A BaMC works collaboratively with a per patient ward server and relies on the server for many functions, including AaA, WTM, and iMAN functions. Moreover, the UI of BaMC differs significantly from that of iNuC. Building a BaMC by modifying the way modules are integrated in current hardwired version of iNuC shown in Figure 3(a) takes considerably more effort than building it from a workflow-base version of iNuC. Figure 3(b) shows a structure for a workflow-based iNuC. We are implementing this version to run on Microsoft Windows Workflow Foundation (WF) in .NET [16].

Details in Figure 3(b) are not important for our discussion here. It suffices to say that the code developed for the current hardwired version can be reused to provide dynamically linked library functions required by activities and workflows. The behavior of the cart and its interaction with the user (i.e., the UI) depend almost solely on the iNuC state machine workflow shown in the bottom of the figure. We can change an iNuC into a BaMC by replacing the iNuC state machine workflow by a BaMC state machine workflow. When implemented on WF, such reconfiguration can be done dynamically without having to restart the cart. This is indeed a convincing demonstration of the merits of workflow-based design in terms of configurability.

The graphical or textual definition of the state machine workflow of a cart provides a clear specification of the cart behavior. We can use it as such to simulate the cart and user-cart interactions in order to assess the usability, correctness and performance of the cart. The simulation environment described in [27] is for this purpose. We have used this approach to validate and evaluate the design of BaMC and the interaction of the cart with the server and the user. Finally, the service interfaces supported by WF makes the dependencies between components more explicit. These advantages are important.

WF has many shortcomings for applications such as iNuC and Pygmalion, however. Running in .NET environment means not only large system resource demands, but also less than ideal response times for time critical functions. To provide real-time scheduling capabilities requires replacing the default scheduling service by a custom one. This can be done in principle, but a custom scheduler in .NET environment is unlikely to give the developer control on scheduling and memory management to the degree necessary for many real-time applications. By supporting state machine workflows as well as event-driven sequential workflow and service interfaces for workflow communications, EMWF 2.0 aims to provide the advantages of WF for embedded applications without its shortcomings.

3 EMWF 1.0 OVERVIEW

This section presents an overview of the design and structures of EMWF 1.0. It also discusses the relative performance of alternative engine architectures.

(A) SISARL-XPDL

The embedded workflow definition language supported by EMWF 1.0 is called SISARL-XPDL. It is an extended subset of the WfMC standard XPDL 2.0 [17]. Table 2 gives examples of SISARL-XPDL elements. How the elements are used can be deduced by and large from their names. We will explain the less obvious ones as needed.

Table 2 Example of SISARL-XPDL elements

Workflow Data	TypeDeclaration, DataType, BasicType, SchemaType, DeclaredType, RecordType, DataField, InitialValue
Workflow Structure	Package, Pool, Lane, WorkflowProcess, BlockActivity, Activity, SubFlow, Task, Application, Implementation
Workflow Control	Transitions, TransitionRef, TransitionRestriction, Route, Merge, Split, StartEvent, EndEvent, Condition, IntermediateEventTrigger, Deadline, Priority
Resources	ParticipantType, Participant, Application FormalParameter, ActualParameter
Extensions	Period, ExtendedAttributes, BCA

The elements listed in the first four groups labeled workflow data, structure, control and resources are elements from the standard XPDL 2.0. The elements in the group labeled extensions are not part of the XPDL. BCA are built-ins activities for behavior coordination, including the ones listed in a bottom row in Table 1. Period and ExtendedAttributes enables the

developer to specify the execution rates and other attributes of activities and workflows. In particular, the developer can inform the workflow manager which workflows are real-time. By default, real-time workflows are scheduled on rate-monotonic basis according to their periods. The developer can override this commonly used scheduling strategy and tell the workflow manager how workflows and activities are to be scheduled via `ExtendedAttributes`.

The SISARL-XPDL parser first translates extension elements into standard XPDL 2.0 elements and then compiles XPDL 2.0 definitions of workflows directly into executable workflow scripts. The script is not easily human readable and error-prone to revise. This shortcoming will be eliminated in EMWF 2.0.

(B) Engine Manager

Figure 4 shows the general structure of a workflow-based devices and major components (i.e., engine manager, workflow manager and workflow processor) of the EMWF 1.0. The XPDL term *participant* in the callout at upper right corner refers to a resource that is managed and allocated to workflows by the engine. A participant may be a hardware device (e.g., bar-code scanner, motor, sensor and even a CPU), a person, and so on. When assigned to do the work, such participants carry out external activities. Participants required by software activities include library functions, interfaces and shared resources.

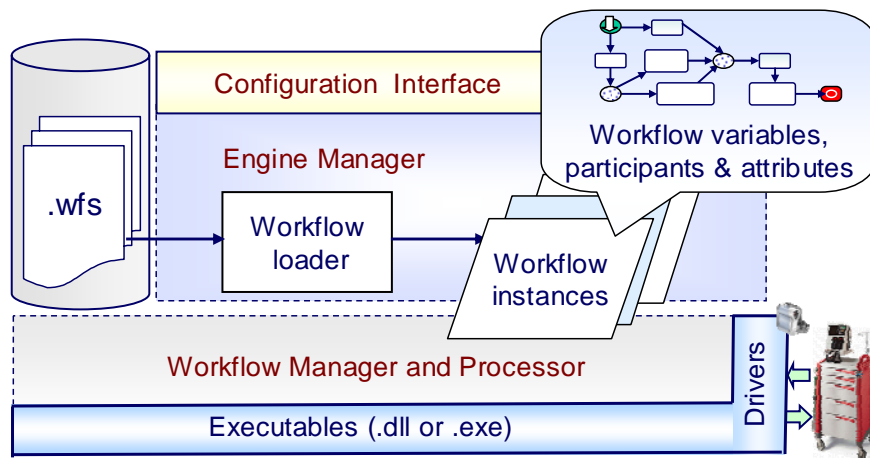


Figure 4 Workflow-based device structure

The engine manager manages the configurations of the engine and application workflows. It is also responsible for handling user requests and managing their accesses to workflow-related definitions and optional contextual information. EMWF 1.0 offers the developer the ability to tune the engine via the configuration interface by changing its *configuration parameters*, which

include the maximum numbers of threads and priority levels, and the finest resolution of timers. The figure shows the configuration on a target device: Workflow definitions have been parsed on a development machine and the resultant workflow scripts are stored in .wfs files. During initialization, the engine manager initializes and configures the engine. It then loads all the .wfs files of the applications into memory.

(C) Workflow Manager and Processor

The workflow manager processes the workflow scripts, and the workflow processor executes activities according to the scripts. To keep the run-time overhead low, sometimes at the expense of memory footprint, the engine manager loads .wfs files of all workflows needed for all operation modes and adaptation during initialization. This allows the workflow manager to dynamically allocate memory for all instances of activities and workflows during initialization. Consequently, although EMWF 1.0 allows .wfs files to be added and removed for configuration purpose, the engine must be restarted for the configuration changes to take effect.

EMWF 1.0 offers two different multi-threaded workflow processors on Microsoft Windows CE and XP Embedded. They are the WLA (workflow-level assignment) engine and the ALA (activity-level assignment) engine. The structures of the engines are shown in Figures 5 and 6, respectively. The term *general activity* in the figure refers to activities provided by the developer, i.e., not built-ins with the engine.

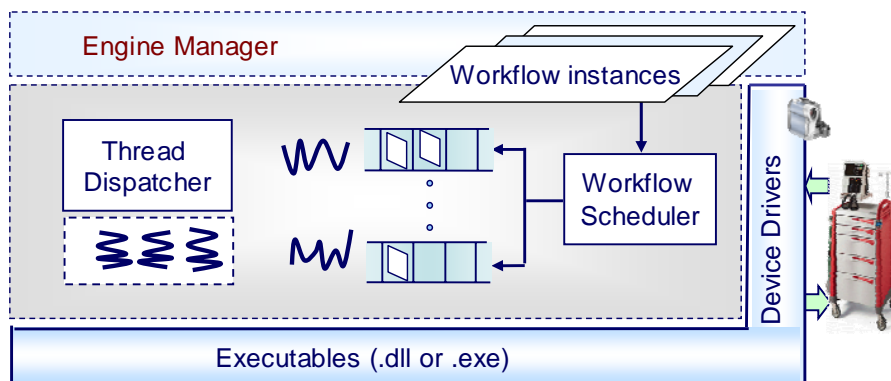


Figure 5 WLA engine structure

In a WLA (workflow-level assignment) engine, shown in Figure 5, each thread is dedicated to a workflow. Specifically, the workflow manager attaches a thread to each workflow when it initializes the workflow and schedules the thread to execute all activities in the workflow. The thread inherits the priority of the workflow. Depending on the engine configuration and current

workload, the manager may assign additional threads to the workflow when a workflow has concurrently executable paths.

In an ALA (activity-level assignment) engine, shown in Figure 6, activities are executed as work items by worker threads: The workflow manager maintains a FIFO queue per priority for queuing work items and assigns at least a thread per queue. The thread (or threads) serving a queue executes at the priority of the queue. Threads in the workflow manager and processor interact in more or less in the leader/followers pattern [33]. Worker threads are followers. For a device that has no blocking built-in activities, the workflow manager may have just one leader thread. The leader processes workflow scripts, wraps ready (i.e., enabled) general activities as work items and inserts them in priority queues according to their priorities to be executed by follower threads, and supervises their completion. With a few exceptions, built-ins are simple. The leader executes itself built-ins as they become enabled, which in turn leads to more general activities be ready and queued. These functions of the leader are depicted as general activity scheduler and built-in activity accelerator in Figure 6.

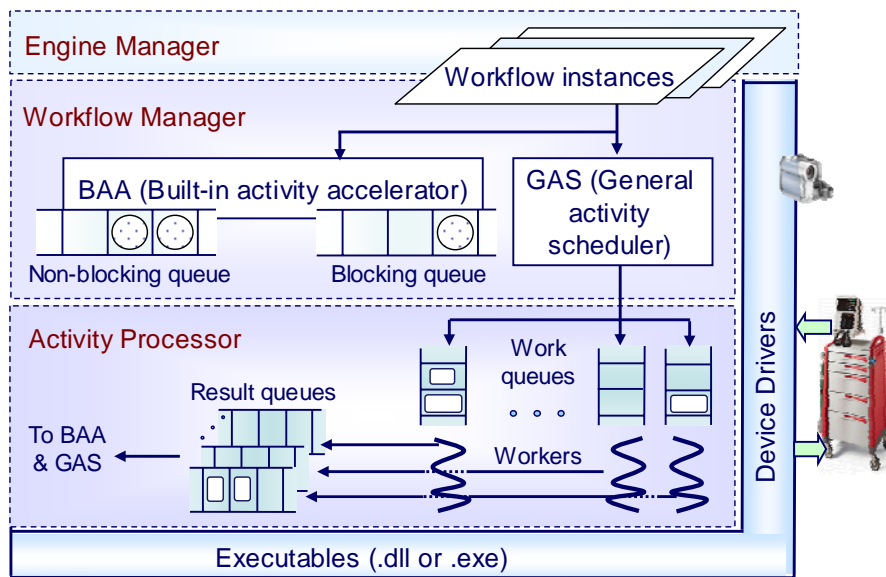


Figure 6 ALA engine structure

(D) Relative Merits

Since a thread assigned to a workflow in a WLA engine executes both general activities and built-ins in it, most of the transitions between activities incur no context switch. It is expensive for threads to change priority. This is why the current versions of WLA engines do not support varying priority within workflows, just like WF, which also uses the WLA strategy. In contrast,

varying priority in a workflow can be done by an ALA engine at no cost. Coherent time order for all tasks within a device is often expensive to achieve. Using an ALA engine, however, this can be accomplished naturally with no additional cost by having the engine use a single thread to handle all timing events. These advantages of ALA are important, especially for devices with time-critical tasks. On the other hand, every transition from one general activity to another incurs at least one context switch. This is a disadvantage when compared with WLA engines.

We have measured the runtime performance of WLA and ALA engines using several benchmark workflow-based workloads running on a 3.4 GHz Pentium 4 and Windows CE 6.0 platform. Each workflow-based workload is characterized by the number of workflows in the load, a test pattern and the granularity of activities in the workflows. For each workflow-based load, we also ran equivalent customized, hardwired code on Windows CE 6.0 without the engine. The difference between the response times for the two versions gives us an estimate of the runtime overhead introduced by the engines.

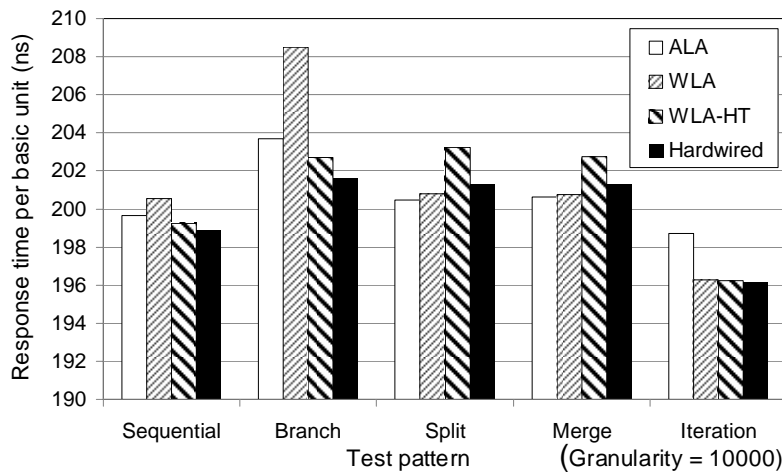


Figure 7 Performance data from [21]

Figure 7 illustrates the kind of performance data obtained from this study. In this case, all activities are software activities with the same granularity. (The function for each activity calls a random number generator 10,000 times.) The WLA-HT engine is an enhanced WLA engine: A helper thread is used to create workflow instances on behalf of worker threads, hence the name WLA-HT. We can see that when activities are sufficient complex and hence are of sufficiently large granularity, as in the case shown here, runtime overheads of ALA and WLA-HT engines are acceptable, though not negligible. As expected, the disadvantage of ALA engine in terms of context switches becomes evident when activities are so small that the number of extra context switches is in order of 1000 per workflow. Details on test patterns, workflow granularities and

measurements taken, as well as additional plots of response times as functions of test pattern and granularity can be found in [21]. Similar measurement performed in Windows XP Embedded points to similar conclusions.

Finally, to be sure that memory footprint is not an issue for majority of SISARL devices, we ran Roomba workflows shown in Figure 1 on the ALA engine. The engine consumes approximately 524 KB after it starts. It consumes 1114 KB after Roomba .wfs file is loaded. Footprint of this order is but a small fraction of available memory for typical service robots.

4 EXTENSIONS IN EMWF 2.0

We discussed earlier rationales for two kinds of extensions to be included in EMWF 2.0. First, messaging passing and end-to-end scheduling capabilities are essential parts of end-to-end workflow management for time-critical distributed and networked applications. These extensions have already been added to EMWF 1.0 on Windows XP Embedded and are described below

The second kinds of extensions include capabilities and tools that are needed to make EMWF 2.0 not only a middleware ideally suited for devices ranging from simple devices such as automatic vacuum cleaner to service robots and automation tools such as Pygmalion and iNuC, but also an excellent design, development and evaluation environment for them. Among these extensions, supports for workflow-to-workflow communication and interactions are most essential. We will describe a preliminary design at the end of this section.

(A) Messaging Mechanism

Again, EWMF 2.0 aims to be an end-to-end real-time workflow management system. Figure 8 illustrates the environment provided by such a middleware as seen by a distributed workflow-based embedded application. In the figure, an agent refers to a software workflow that executes on a processor on behalf of a remote component. Filled triangles represent Send and Receive. They are the underlying operations for push data and pull data built-in activities, respectively. Unfilled triangles represent events. Events are routed by a system of service interfaces. These interfaces are primary means of communication and interaction among workflows. They are similar to local services provided by WF [16]. We will return shortly to describe the differences between EMWF service interfaces and WF local services.

Figure 9 shows an example. We use it to help us explain how send and receive messages/data are implemented. The example contains two workflows. Workflow 1 contains a push data built-in while Workflow 2 contains two pull data built-ins.

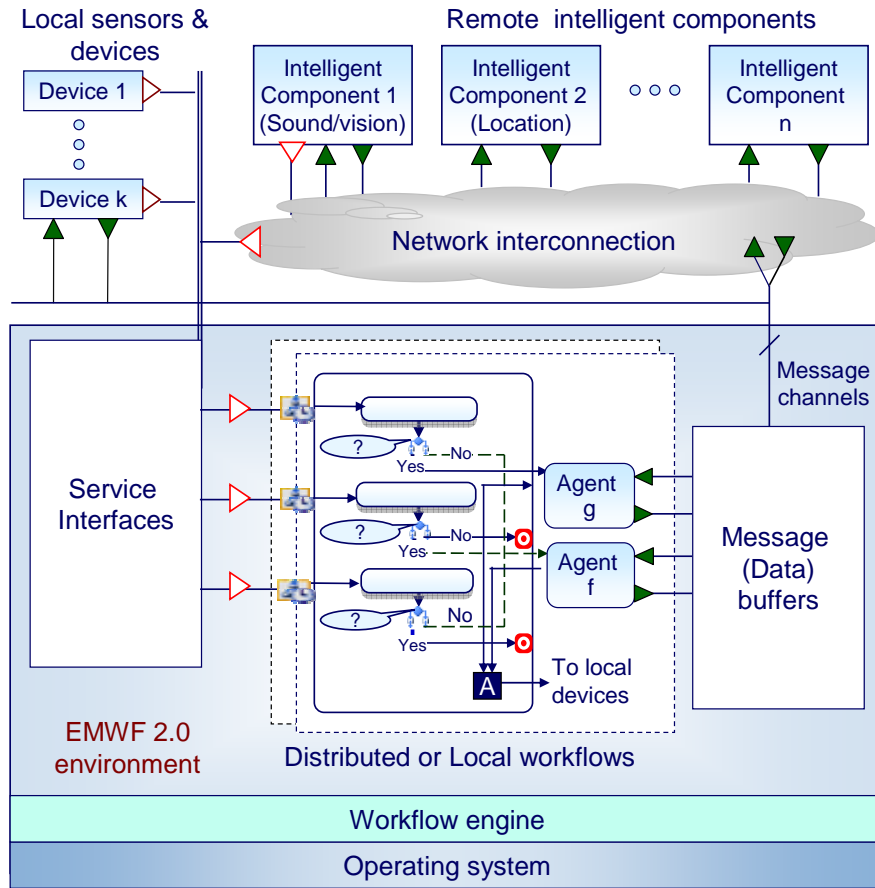


Figure 8 End-to-end workflow management for distributed applications

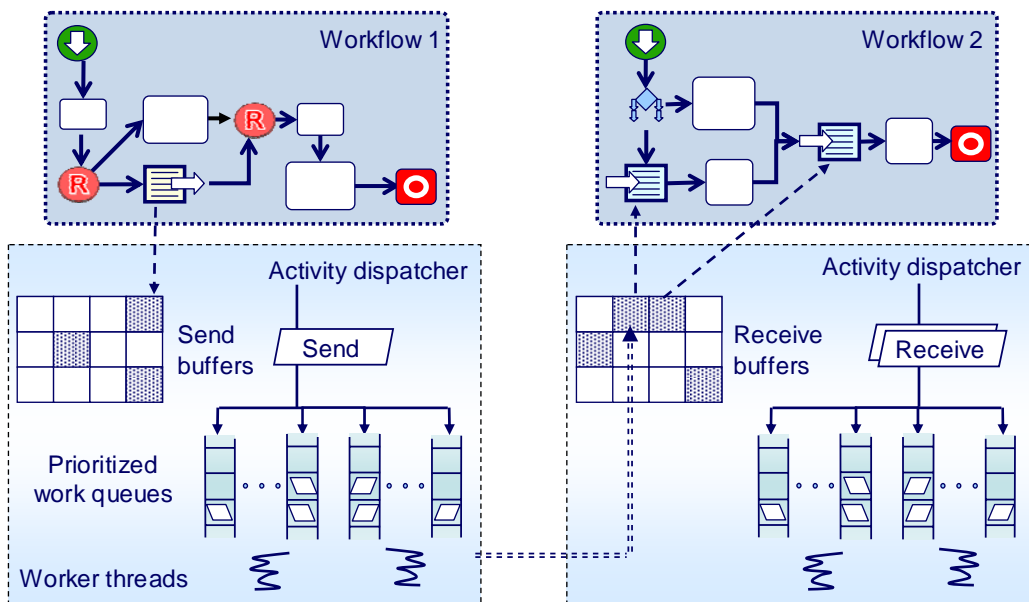


Figure 9 Send/Receive under the control of a message scheduler

When encountering a push data built-in activity, the workflow manager dispatches a `Send` operation to move the data to be sent to Workflow 2 into a send buffer and queues a `Send` work item in one of the prioritized work queues. The work item is then executed by a worker thread according to the priority of the `Send` activity. Depending on whether the receiving workflow (e.g., Workflow 2 in Figure 9) runs locally or remotely, the `Send` work item either invokes an IPC or a Winsock send API function to move the data from the send buffer to the receive buffer. This and other data transfers are depicted by dashed arrows in the figure.

Once arrived, the data waits in the receive buffer until the workflow manager of the receiving workflow encounters a pull data activity, depicted as a box with a block arrow pointing into the box. The manager then queues a receive work item to move the data from the receive buffer to the space of the receiving workflow. Specific work to be done by `Receive` is application dependent. The developer can specify it is to be done via a parameter of `Receive`.

(B) End-to-end Scheduling

The end-to-end scheduler implemented as a part of real-time extension of EMWF 1.0 makes two assumptions on execution model and configuration. First, the system is statically configured: Here, the term static configuration means specifically that components of each application are partitioned and assigned to processors (and other types of resources) at initialization time. Components execute on the processors assigned to them. They are migrated to other processors only when reconfiguration becomes necessary. In contrast, in a dynamically configured system, the scheduler selects a processor for each execution instance of each component based on some criterion (e.g. processor load). The static configuration assumption is restrictive only when the platform contains multiple identical processors (e.g., as in a SMP or with replicated network connections). Advocates of the dynamic alternatives have been able to demonstrate higher processor utilization. This gain is at the expense of considerable increase in complexity of the scheduler. More seriously, there is yet no reliable and efficient ways to validate the timeliness of dynamic systems. For typical SISARL applications, the advantages of static configuration outweigh its disadvantage in processor utilization.

The second assumption is that the distributed nature of the applications is explicit. The validity of this assumption follows naturally in case of statically configured system. The scheduler(s) must have the information on the locations and connectivity of all components in order to schedule them effectively. At runtime, this information is made available to the scheduler(s) via the values of participants (i.e., required resources) and extended attributes of workflows. These values may be specified by the developer of the application when the

developer chooses to partition the application and assign processors/resources to individual partitions manually. EMWF 1.0 allows only this option. EMWF 2.0 will provide or integrate with a resource management tool separate from the scheduler to support this function. For real-time applications, the tool also perform admission control (i.e., allowing a new real-time application to start only when the system can meet all hard real-time requirements of the application in midst of applications that have already started.)

To be concrete, Figure 10 shows the structure of a two-level scheduler for ALA engine. The descriptions of its operations below in term of this structure are by and large applicable to the case of WLA engine.

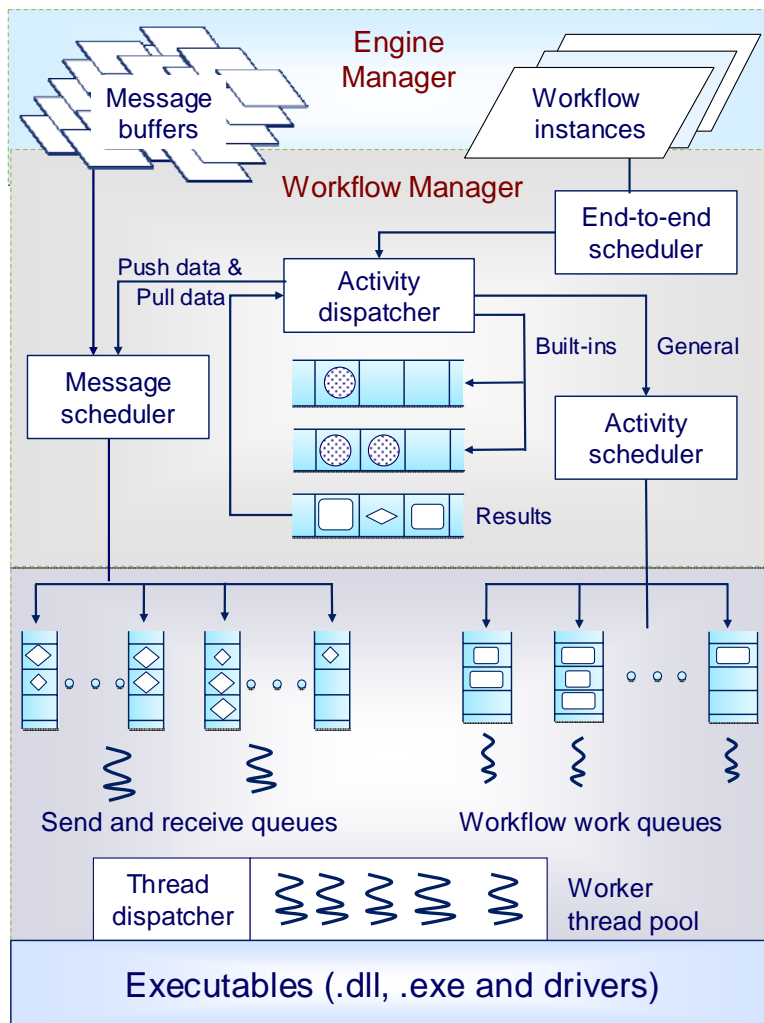


Figure 10 Two-level scheduler structure for ALA engine

Because of the assumptions stated above, the extended attributes of each workflow instance contains information on the CPU used for each software activity and resources required by

external activities. If the workflow has real-time requirements, the extended attributes also specify a finite end-to-end deadline, estimated worst-case execution time of each activity and so on. The *end-to-end slack time* of the workflow is the difference between its end-to-end deadline and the total worst-case execution time of the longest chain of activities in the workflow.

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 to activities or sub-chains of activities. To explain, let us use the application in Figure 10 as an example. Suppose that the workflows in it run on two CPUs connected via a LAN. The longest chain of activities consists of three sub-chains: The first one is from the start of Workflow 1 to push data built-in; the second one contains the send/receive operations over the network; the third sub-chain starts from the first pull data built-in to the end of the Workflow 2. Suppose that the end-to-end deadline is one second, the total worst-case execution times of software activity chains on both CPUs are equal to 100 ms, and worst-case time required to transmit data (when there is no contenting traffic) is 200 ms. This means that the end-to-end slack time for this application is 600 ms. Algorithms for end-to-end scheduling range from simple ones that simply divide the end-to-end slack time evenly (i.e., 200 ms each) or in proportion to their total worst-case execution times (i.e., 150 ms, 300 ms, and 150 ms) to complicated schemes make use of the load conditions on the CPUs and the network, costs of communication between processors, etc. Depending on the algorithms chosen by the developer when the workflow management system is configured, the end-to-end scheduler does this division accordingly.

As Figure 10 shows, both low-level schedulers (activity scheduler for CPU scheduling) and message scheduler (for scheduling network traffic) support fixed priority scheduling. Given the slack times of individual activities or sub-chains, the low-level activity scheduler and message scheduler may compute the relative deadlines of the activities or sub-chains and assign priorities to them accordingly. The schedulers may intentionally hold ready activities for some time before queuing them. The delay thus introduced is called release guard [32]. This action delays the execution of some activities in order to improve the response times of others. The two-level scheduler in the EMWF real-time extension will enable us to experiment with these schemes.

(C) Service Interfaces

We introduced the term Service Interfaces earlier in Figures 3 and 8 without explaining what it means, what the components bearing this name do and why EWMF 2.0 needs to support

them. Simply put, from the point of view of a workflow-based application, each service interface (e.g., StateMachine or GenericComm interface in Figure 3) is a set of interface functions defined by the developer of the application. Workflow instances in it communicate with non-workflow component(s) and with each other using these interface functions.

Service interfaces resemble closely local services supported by Window WF. The kind of assistant EMWF 2.0 will provide to service interfaces is similar to what Window WF does for local services [16]: In particular, similar to WF local services, 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 11 illustrates ways workflow instances and non-workflow components communicate and interact making use of service interfaces. Rectangular shapes represent service interfaces, host applications and non-workflow components. Polygons represent workflow instances of the same or different type. Bold arrows represent callbacks while arrows of various line styles represent raises and deliveries of events.

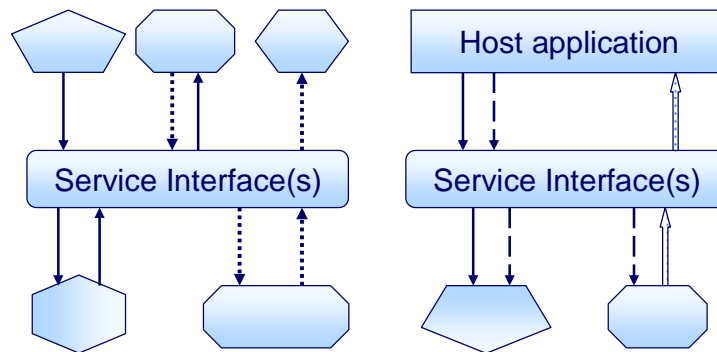


Figure 11 Interaction via service interfaces

The left half of the figure says that 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 helps to track the locations of all workflow instances and thus relieves the developer from the burden of this work.

The right half of Figure 11 says that in addition to events, workflow instances can communicate with the host application and other non-workflow components via callbacks. This is indeed how workflow instances deliver results to UI in iNuC. (This single-threaded UI is the host. It creates and starts the workflow engine, creates and registers local services and creates and starts all workflow instances during initialization).

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. This is the case of the host in iNuC. The host application contains a single thread. After initialization, it becomes the UI thread. Using the communication pattern depicted by Figure 11, the UI thread is never tied up doing time consuming work. By doing so, it stays free and ready to respond to user action.

5 SUMMARY

This report describes the current status and future plans for EMWF. The workflow management system is specifically for embedded applications, ranging from small housekeeping devices like automatic vacuum cleaners to fairly complex devices and systems like iNuC, service robots and other personal and home automation devices. The middleware enables such embedded devices to be built from activities and workflows components. Its light-weight engine integrates the components at runtime by executing the components in manners specified by the developer. Configurability is one of primary factors that motivated us to build EMWF. Illustrative examples in previous sections and case studies done today indeed demonstrate this merit

EMWF 1.0, the current version of the embedded workflow framework, provides light-weight engines on Microsoft Windows CE and XP Embedded platforms. A Linux engine is nearly complete. Workflows defined in terms of the small but extensible language SISARL-XPDL is first translated into standard XPDL and then compiled into binary, executable workflow scripts.

EMWF 1.0 is released under GPL license. Being a proof-of-concept prototype, this version has many limitations. We are working to remove them while adding features and capabilities and maturing the middleware into EMWF 2.0. Below are examples of on-going and future work towards this goal.

- EMWF 1.0 is intended for small devices in which all software components fit on a processor. Anxious to minimize runtime overhead and keep memory footprint reasonably

small, we made some code design choices that limit configuration and customization be done at compile and build times, but not at runtime. We are now working to remove this limitation without sacrificing runtime performance.

- EMWF 1.0 has no messaging facility needed to support push and pull data built-ins. While it provides the mechanism needed to support priority-driven CPU scheduling strategies, it lacks supports for message scheduling and end-to-end scheduling, resource allocation and admission control capabilities needed by time-critical embedded applications that run on multiple CPUs and microcontrollers and over various networks. Extensions needed to remove some of these limitations are now available in the Window XP Embedded engine.
- EMWF 1.0 together with the above mentioned extensions are necessary but not sufficient for end-to-end workflow management of applications that have many modules, may have real-time requirements, and need to evolve and change over their lifecycle. Services that support workflow to workflow communication and interaction and their interaction with the host application are essential and will be supported by EMWF 2.0. Their design and implementation are among our top-priority future tasks.
- We also want to extend the SISARL-XPDL parser to produce intermediate workflow scripts in C with annotations and extensions. The SISARL-XPDL parser now translates workflows definitions directly into binary workflow scripts executable by EWMF engines. For relative simple applications, this suffices. We anticipate, however, that workflow scripts in C-like programming language can give developers of large complex workflow-based applications an added convenience: During debugging, experimentation and evaluation, the developer will have the choices between using SISARL-XPDL, taking advantage of the XPDL 2.0 GUI editor, or directly program the workflow in C programming language.

6 ACKNOWLEDGMENT

This work is partially supported by the Taiwan Academia Sinica thematic project SISARL (Sensor Information Systems for Active Retirees and Assisted Living), as well as funds provided by CITI (Center for Information Technology Innovation), Academia Sinica and ITRI (Industrial Technology Research Institute).

7 REFERENCES

- [1] Wang, W. Y., J. K. Zao, P. H. Tsai, and J. W. S. Liu, "Wedjat: A Mobile Phone Based Medication Reminder and Monitor," *Proceedings of the 9th IEEE International Conference on Bioinformatics and Bioengineering*, June 2009.
- [2] Tsai, P. H., C. Y. Yu, C. S. Shih and J. W. S. Liu, "Smart Medication Dispenser: Architecture, Design and Implementation," Technical Report No. TR-IIS-008-010, Institute of Information Science, Academia Sinica, 2008.
- [3] Liu, J. W. S., C. S. Shih, P. H. Tsai, H. C. Yeh, P. C. Hsiu, C. Y. Yu, and W. H. Chang, "Point-of-Care Support for Error Free Medication Process," *Proceedings of High-Confidence Medication Device Software and Systems Workshop*, June 2007
- [4] Chou, T. S. and J. W. S. Liu, "Design and Implementation of RFID-Based Object Locator," *Proceedings of IEEE 2007 International Conference on RFID Technology*, March 2007.
- [5] Tsai, P. H., H. C. Yeh, C. Y. Yu, P. C. Hsiu, P. C. S. Shih and J. W. S. Liu, "Compliance Enforcement of Temporal and Dosage Constraints," *Proceedings of IEEE Real-Time Systems Symposium*, December 2006.
- [6] Hsu, Y., S. F. Hsiao, 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 Systems," *Proceedings of IEEE International Conference on SMC*, September 2006
- [7] Hsu, C. F., H. Y. M. Liao, P. C. Hsiu, C. S. Shih, T. W. Kuo, and J. W. S. Liu, "Smart Pantries for Homes," *Proceedings of IEEE International Conference on SMC*, Sept. 2006
- [8] Yeh, H. C., P. C. Hsiu, C. S. Shih, P. H. Tsai, and J. W. S. Liu, "APAMAT: A Prescription Algebra for Medication Authoring Tool," *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, September 2006.
- [9] Forizzi, J. and C. DiSalvo, "Service Robots in Domestic Environment: a Study of Roomba Vacuum in the Home," *Proc. of ACM/IEEE International Conference on HRI*, March 2006.
- [10] Kaneshige, Y., M. Nihei, and M. G. Fujie, "Development of New Mobility Assistive Robot for Elderly People with Body Functional Control," *Proceedings of IEEE/RAS-EMBS*, February 2006.
- [11] Lin, C. H., Y. Q. Wang and K. T. Song, "Personal Assistant Robot," *Proceedings of IEEE International Conference on Mechatronics*, July 2005.
- [12] Tsai, P. H., Y. T. Chuang, T. S. Chou, C. S. Shih, and J. W. S. Liu, "iNuC: An Intelligent Mobile Medication Cart," *Proceedings of the 2nd International Conference on Biomedical Engineering and Informatics*, October 2009.

- [13] “Speci-Minder Autonomous Hospital Robots,” <http://robots.net/article/2156.html> , 2007
- [14] <http://www.informatics.nhs.uk/cgi-bin/item.cgi?id=1155> , “Washington hospital implements drug dispensing robots,” February 2005.
- [15] Workflow definition, <http://en.wikipedia.org/wiki/Workflow>
- [16] Bukovics, B. Pro WF: Windows Workflow Foundation in .Net 4.0, Apress 2009.
- [17] WfMC: Workflow Management Coalition, <http://www.wfmc.org/> and WfMOpen, <http://wfmpen.sourceforge.net/>.
- [18] XPDL (XML Process Definition Language) Document, http://www.wfmc.org/standards/docs/TC-1025_xpdl.2.2005-10-03.pdf, October 2005
- [19] BPEL (Business Process Execution Language), <http://en.wikipedia.org/wiki/BPEL>
- [20] Open Source Java XPDL editor, <http://www.enhydra.org/workflow/jawe/index.html>.
- [21] Chou, T. S., 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,” *Proceedings of IEEE Real-Time and Embedded Applications and Systems Symposium*, April 2009.
- [22] Enhydra Shark, <http://forge.objectweb.org/projects/shark>
- [23] Pajunen, L. and S. Chande, “Developing workflow engine for mobile devices,” *Proceedings of IEEE International Enterprise Distributed Object Computing Conference*, 2007.
- [24] Hackmann, G., M. Haitjema, C. Gill, and G. C. Roman, “Silver: A BPEL workflow process execution engine for mobile devices,” in A. Dan and W. Lamersdorf, Ed., *ICSOC 2006*.
- [25] Jing, J., K. Huff, B. Hurwitz, H. Sinha, B. Robinson, and M. Feblowitz, “WHAM: supporting mobile workforce and applications in workflow environments,” *Proceedings of the 10th IEEE Workshop on Research Issues in Data Engineering*, February 2000.
- [26] 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.
- [27] 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.
- [28] S.-Y. Chang, Y.-F. Lu, T. W. Kuo, and J. W. S. Liu, “The Design of a Light-Weight Workflow Engine for Embedded Systems,” *Proceedings of RTSS Workshop on Software and Systems for Medical Devices and Services*, December 2007.

- [29] Brega, R., N. Tomatis, K. O. Arras, “The Need for Autonomy and Real-Time in Mobile Robotics: A Case Study for X0/2 and Pygmalion,” *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2000.
- [30] Robot Standards and Reference Architecture,
<http://wiki.robot-standards.org/index.php/Middleware>
- [31] D. Hull, A. Shankar, K. Nahrstedt, and J. W. S. Liu, “An end-to-end QoS management architecture,” *Proceedings of IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, December 1997.
- [32] J. Sun, R. Bettati and J. W. S. Liu, “An end-to-end approach to schedule tasks with shared resources in multiprocessor systems,” *Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software*, May 1994.
- [33] Schmidt, D. C. et al., “Leader/followers: a design pattern for efficient multithreaded event de-multiplexing and dispatching,”
<http://ftp.icm.edu.pl/packages/ace/ACE/PDF/lf-PLOPD.pdf>
- [33] SISARL (Sensor Information Systems for Active Retirees and Assisted Living),
<http://sisarl.org>