A Mobile Agent Approach to Lightweight Process Workflow

Gail Kaiser, Adam Stone and Stephen Dossick

Columbia University
Department of Computer Science
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
United States

212-939-7100/fax:212-939-7084
psl@cs.columbia.edu
http://www.psl.cs.columbia.edu/

Introduction

The Programming Systems Lab at Columbia University has investigated software process modeling and enactment since its inception in the mid-1980s, initially in the **Marvel** project [1,2]. In the early to mid-90s, we extended to cross-organizational processes operating over the Internet, in **Oz** [3,4] and **OzWeb** [5]. That is, Oz enabled the software development team and other stakeholders to be geographically, temporally and/or organizationally dispersed. OzWeb added integration of Web and other external information resources whereas Oz and Marvel had assumed all project materials to be resident in their native objectbases. OzWeb's plugin services and tools were accessible via conventional Web browsers, HTTP proxies and Java GUIs, improving dramatically on Marvel's and Oz's X11 Windows XView/Motif user interface clients. The successive prototype frameworks we developed and demonstrated were used on a daily basis in-house to maintain, deploy and monitor their own components, APIs and user interfaces.

Novel (at the relevant time) framework components included rule-based process modeling and a corresponding enactment engine supporting multi-process interoperability for joint and subcontracted tasks across autonomous organizations (**Amber**); a transaction monitor customizable to application-specific long duration and group extended transaction models over Web and legacy resources (**Pern**, succeeded by **JPernLite**); an object manager supporting multi-inheritance (**Darkover**); a decentralized intranet-remote tool-sharing service that turned legacy only-executables-available single-user tools into groupware (Multi-Tool Protocol or **MTP**, later **Rivendell**); an OQL and XML-based "light semantics" information integration service that could impose external hyperlinks and annotations (**Xanth**); an integrated object broker/events messenger that enabled dynamic integration and publish/subscribe for legacy and new data and computation services (**Groupspace Controller**); a service for inserting HTML links in both directions between identifier uses and definitions for any source code file sets supported by the etags utility (**Hi-C**); a RVP-style instant messaging system (Java Instant Messaging or **JIM**); and a toolkit for programming smart federations among enterprise-enabled services (**TreatyMaker**). [Citations/References intentionally omitted from this paragraph due to space considerations; see ftp://ftp.psl.cs.columbia.edu/pub/psl/INDEX.html.]

The new process technology first presented here is broadly based on our decade of research on and experimentation with architecting and using such prototype services and software development processes targeted to Internet/Web middleware and applications, but reflects a major departure from our own (and others') previous directions. In particular, current process and workflow systems, including our own, are often too rigid for open-ended creative intellectual work, unable to rapidly adapt

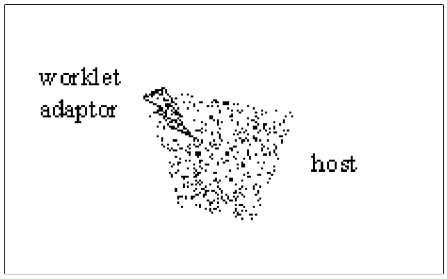
either the models or the enactment to situational context and/or user role. On the other hand, the process/workflow ideal implies a flexible mechanism for composition and coordination of information system components. We now present our in-progress development of rehostable lightweight mobile agents for on-the-fly process construction, adaptation and evolution, system reconfiguration, and knowledge propagation.

Worklets Model

Scripted mobile workflow agents, which we call *worklets*, address both the problems and the promise: Worklets might be constructed or parametrized on the fly by a human or a program, then transmitted from component host to host through a "meta-workflow" - a dynamically determined routing pattern reactive to the latest host's circumstances and surroundings as well as past and planned trajectories. Workflow typically involves actions performed on data, or perhaps interactions among humans concerned with implicit data "resident" in the humans' memories. But here the "work" generalizes to (re)customizing the host's configuration - loosely construed, including, e.g., schemata, lock tables, authorization capabilities, event subscriptions, even host machine registry. And of course the process model(s). In the degenerate case of the usual data, a worklet is simply a workflow snippet whose semantics are dependent on the host's interpretation of its directives. [Note that by host we generally mean a particular information system component, not usually the entire machine or operating system platform.]

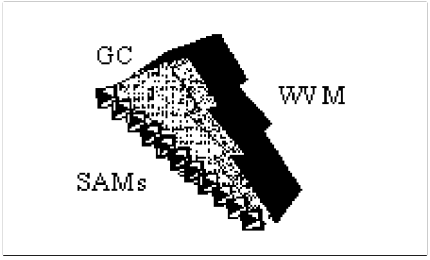
Each worklet is a small scripted program, like the various web agents (e.g., see [6]) a combination mobile agent and smart RPC, but in our case potentially including workflow-like rules, as well as imperative code for host-context exploration/instantiation, whose "*work*" is to manipulate the configuration model(s) of a middleware service or a complex document. The level of dynamism is inherently both enabled and limited by the host: For example, in the case where the host is a database management system and the worklet initiates changes to its schema, that "(re)configuration" might immediately evolve all data, upgrade data as it happens to be accessed, apply only to new data, or become effective only after a long off-line procedure. However, the "configuration" implied by the database's contents could usually be modified on the fly as worklets arrive or the triggering conditions of already-local worklets become satisfied. As another example, worklets might define part of or modify the workflow definition being enacted by a conventional workflow management tool, inserting their bodies into the model or matching against existing tasks to be adapted or removed. Whether or not a newly modified process model applies to any in-progress process steps, the current or following spiral iteration, or only to the "next" instance is necessarily limited by the capabilities of the base workflow management system. Unless, of course, the worklet enacts a workflow fragment on its own. Any part of an intelligent document could be treated as a configuration model to be upgraded by the worklet, e.g., to tailor and install its components in a distributed enterprise setting.

Worklets Architecture



A host-specific worklet adaptor must be constructed for each anticipated host system or component,

and is attached to that host, as illustrated above. Obviously, construction of such adaptors is plausible only if the host provides an API or extension language, can reasonably be wrapped, or of course if its source code is available and the adaptor builder is willing and able to plunge into it. Generally, the adaptor builder must have expert-level understanding of the host and the capabilities it exports. However, the worklet writer should have no need to understand any particular host, and usually worklets should be written without any particular host(s) in mind.



Worklets are intended to be interpreted by a common *worklet virtual machine* (WVM) embedded in the host-specific worklet adaptor, as shown above. The adaptor translates those internal configuration capabilities that the host exposes into terminology meaningful to the worklets through *service access modules* (SAMs). Of course, some worklet directives may be impossible to map to anything other than a no-op: a worklet intent on writing to a console will obviously not do anything very useful if it lands on a device with no user interface and no file or other storage capacity to approximate an output stream.

SAMs are connected to WVM through our *groupspace controller* (GC). GC was initially designed to support workgroup information spaces, by imposing what we termed *groupspace services* on all data and computation accesses/requests [7]. Here, GC hooks into the host's event bus, or closest approximation thereof, to inform WVM of any internal events of interest and to publish to WVM in a standard manner those host functions available through the SAMs. Although it is possible to implement the host interface as a single SAM, generally the host's features are divided into functionally cohesive "services", and thus multiple SAMs.

Worklets Realization

The Worklet Virtual Machine is responsible for maintaining an execution environment for the worklets in a given system. This includes maintaining worklet interpreters and the threads they run in, as well as an extensible network layer for transporting worklets between WVMs and serialization (what Python calls "pickling") capabilities. WVM is implemented in Java 2, and the worklets themselves are written in JPython. Worklets are transmitted among hosts and communicate with each other through Java RMI. SAMs should be written following the JavaBeans convention, and their interfaces are also published as JavaBeans. The **Listener** worklet below simply listens to events from GC and "handles" them. **MyWorklet** immediately sends itself to another host and starts emitting events from there.

```
class ListenerWorklet(Worklet):
    def foobar(self):
        print "Inside foobar"
        print interp.getEnvironmentName()
    def activate(self):
        print "In activate"
        interp.addSubscription( "WVM.eventGenerator.EVENT" )
        interp.enterEventLoop()
    def handleEvent(self):
        print "Handling Event"
        print __currentEvent.getEventDescription()
```

```
            print "Done Handling, thanks..."
    w = ListenerWorklet()

    class MyWorklet(Worklet):
        myCount = 1
        def foobar(self):
            print "Inside foobar"
            print interp.getEnvironmentName()
        def activate(self):
            if (self.myCount == 1) :
               self.myCount = 0
               interp.sendCopy( "broadway" )
            else :
               print "Worklet arrived safely"
               print "Sending events"
               interp.sendEvent( "EVENT" )
               interp.sendEvent( "OTHER" )
               interp.sendEvent( "EVENT" )
               print "Done Sending events"
    w = MyWorklet()
```

Applications

Systems constructed using our **CHIME** (Columbia Hypermedia IMmersion Environment [8]) infrastructure present their users with a 3D depiction of hypermedia and/or other information resources. Users visualize, and their avatars operate within, a collaborative virtual environment [9] based on some metaphor selected to aid their intuition in understanding and/or utilizing the information of interest or relevant to the task at hand. Users "see" and interact with each other, when in close [virtual] proximity, as well as with the encompassing information space. Actions meaningful within the metaphor are mapped to operations appropriate for the information domain, such as invoking external tools, running queries or viewing documents. An e-commerce web site peddling computer hardware might look and feel like an on-screen CompUSA; a digital library might be illustrated as, indeed, a library. Application domains without obvious physical counterparts might choose more whimsical themes. For example, a software development environment for an open-source system might map each source code package to a room on the Starship Enterprise, with the "main" subprogram represented by the bridge, amateur programmers proposing a modification could beam aboard, and so forth. Note these are just possibilities: CHIME is a generic architecture, no particular theme is built-in. But environment designers do not necessarily need to program since graphic textures and models can be supplied by third parties, and the specific layout and contents of a world are automatically generated according to an XML-based configuration. The environment designers must, of course, understand their backend repositories sufficiently to write the XML and corresponding processors, unless such meta-information is already supplied by the sources.

CHIME will employ worklets in two ways: Incremental update of the virtual worlds, and lightweight XML processing. When a backend information resource is modified, generally through an external tool, the corresponding specialized kind of SAM called a *data access module* (DAM), generates and emits a worklet to update the internal representation of the virtual world, which is maintained by the Virtual Environment Model (VEM) component of the infrastructure. VEM in turn generates and emits a worklet to the theme manager, to make the corresponding updates there. Finally, the theme manager generates and emits a worklet for each of its active clients. Such worklets are intepreted by each client's *presentation access module* (PAM), another subclass of SAMs, to initiate interpolation from the previous scene graph viewed by that client to the new form (if desired, a user can always choose to stick with an older version). The twin mappings from backend sources to VEM to theme manager are

defined in XML by an environment administrator. There is no predefined DTD covering the tags added at each processing point, but instead unrecognized tags are handled by retrieving the matching worklet from an "XML oracle" component (which can also be employed separately from the rest of the CHIME infrastructure). Both processes are obviously very simple, conceptually, but must be extremely dynamic and flexible, with the constituent workflow tasks effectively conceived and routed on the fly.

Workgroup Cache [10] is intended to reduce average access latency for shared networked documents. *Zero latency* refers to the condition where access to a document produces a cache hit on the local machine, because it was either previously accessed or prefetched; *negative latency* refers to automatic presentation of a document to a user based on a prediction that this user will soon want that document. Simple algorithms like LRU cache replacement and explicit keyword or topic subscriptions are routinely used when nothing is known about the semantic content of the accesses or the tasks to be performed by the user utilizing those accesses. However, our Workgroup Cache infrastructure can (in principle) leverage any knowledge available about the semantic content and pragmatic usage of documents as a basis for prediction of future accesses. We focus that knowledge within a "workgroup", a set of users working on the same task or related tasks, as opposed to the "universal" statistics maintained by Alexa for Netscape's "What's Related". The members of a given workgroup can be explicitly specified in advance, such as a software development team working closely together (although they might be physically dispersed), or determined dynamically by including users whose document accesses match patterns associated in some manner with the workgroup, such as amateur programmers actively working on the same subsystem of an open-source project [11].

Open source development relies on coordinating numerous developers distributed world-wide; difficulties can arise due to the relatively little contact developers may have with each other, they might never even have met! A system for recommending documents (such as source files) based on the nature of their present tasks and the document content might help. For example, say a primary developer is making major changes to one file in a module, and a secondary developer attempts to start adding a small feature to another file in the same module. If both developers are already or automatically become (due to their apparently shared interests) members of the same workgroup, and are using our Workgroup Cache system powered by semantics drawn from software configuration management, the cache system would "know" about the dependencies between the two files, and could push (recommend) the changing source file to the secondary developer. Then she could check if the code she was adding will be incompatible with the likely new version of the module. Or pull criteria might continuously update the feature self-assignments records, so she could also be informed that another secondary developer had already started adding a similar feature, preventing duplication of effort.

Various cache fill (push) and replacement (save) criteria, as well as recommendation criteria (push), may be defined separately as worklets and then associated on the fly with a given workgroup. Criteria might be based on software process or workflow routing among workgroup members, through a conventional process management system that emits "to do" assignments as worklets. Or worklets could directly analyze and respond to document access patterns of workgroup members (e.g., if my supervisor or guru colleague keeps returning to such and such technical report then I want to read it too). Finally, the "XML oracle" can also operate here, to supply worklets that process XML metadata associated with or embedded in accessed documents. Again, the portion of the workflow we anticipate handling through worklets is relatively simple, but necessarily highly dynamic and flexible. Worklets are intended to operate in tandem with conventional process automation and workflow management systems, adapting/evolving their process models as need be, being emitted by local worklet adaptors to perform remote tasks - particularly tasks whose makeup and hosts are determined while the task is in progress, or on small devices unlikely to support a full-fledged workflow management system.

References

#. Gail E. Kaiser and Peter H. Feiler, An Architecture for Intelligent Assistance in Software Development, *9th International Conference on Software Engineering*, March 1987, pp. 180-188. ftp://ftp.psl.cs.columbia.edu/pub/psl/icse87.ps.gz.

#. Israel Z. Ben-Shaul, Gail E. Kaiser and George T. Heineman, An Architecture for Multi-User Software Development Environments, *Computing Systems, The Journal of the USENIX Association*, 6(2):65-103, Spring 1993. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-012-92.ps.Z.

#. Israel Z. Ben-Shaul and Gail E. Kaiser, A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment, *16th International Conference on Software Engineering*, May 1994, pp. 179-188. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-024-93.ps.Z.

#. Israel Z. Ben-Shaul and Gail E. Kaiser, Federating Process-Centered Environments: the Oz Experience", *Automated Software Engineering*, 5(1):97-132, January 1998. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-006-97.ps.gz.

#. Gail E. Kaiser, Stephen E. Dossick, Wenyu Jiang, Jack Jingshuang Yang and Sonny Xi Ye, WWW-based Collaboration Environments with Distributed Tool Services, *World Wide Web*, 1:3-25, 1998. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-003-97.ps.gz.

#. Michael N. Huhns and Munindar P. Singh (eds.), Internet-Based Agents: Applications and Infrastructure, special issue of *IEEE Internet Computing*, 1(4), July/August 1997. http://www.computer.org/internet/ic1997/w4toc.htm.

#. Gail E. Kaiser and Stephen E. Dossick, Workgroup Middleware for Distributed Projects, 7th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, June 1998, pp. 63-68. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-006-98.ps.gz.

#. Stephen E. Dossick and Gail E. Kaiser, CHIME: A Metadata-Based Distributed Software Development Environment, to appear in Joint *7th European Software Engineering Conference* and *7th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, September 1999. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-006-99.zip.

#. Elizabeth F. Churchill and Dave Snowdon, *Report on Collaborative Virtual Environments 1998 (CVE '98)*, June 1998, http://www.fxpal.com/cve98/Report.

#. Gail Kaiser, Christopher Vaill and Stephen Dossick, A Workgroup Model for Smart Pushing and Pulling, to appear in *8th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 1999. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-012-99.zip.

#. Charles C. Mann, Programs to the People, *Technology Review*, January/February 1999. http://www.techreview.com/articles/jan99/mann.htm