Bite: Workflow Composition for the Web

Francisco Curbera, Matthew Duftler, Rania Khalaf, and Douglas Lovell

IBM T.J. Watson Research Center, Hawthorne, NY 10532, USA
{curbera, duftler, rkhalaf, dclo}@us.ibm.com

Abstract. Service composition is core to service oriented architectures. In the Web, mainstream composition is practiced in client-side or server-side mashups, such as providing visual widgets on top of Google Maps results. This paper presents an explicit, workflow based composition model for Web applications called Bite. In contrast with prior attempts to bring workflow capabilities to the Web environment, Bite can deal with data integration as well as interactive, asynchronous workflows with multi-party interactions, and is architected to support protocols currently in use by Web applications. The Bite development model is designed for simplicity and short development cycle by taking a scripting approach to workflow development.

1 Introduction

It is probably fair to say that service oriented architectures [1] deliver two main values: extended interoperability (runtime as well as tools) and service composition. It is hard to argue at this point with the success of the SOC approach, as its wide adoption by enterprises and public organizations demonstrates.

In the last few years, however, questions have been raised from Web-centric developers about the complexity and overhead of the SOA and Web services models [2]. Interoperability, it is argued, was delivered by the Web years ago and at a much lower overhead to both runtime systems and developers. While failing to address the need for end-to-end quality of service and tools in enterprise settings, this argument is certainly appropriate in the context in which it is made: Web application development. This paper is not concerned with this debate, but with the related question of how to bring composition capabilities like those at the heart of SOA to a Web-centric environment.

Composition is of course not new to the Web. The resource oriented architecture of the Web has favored data-centric composition models such as those underlying most "mashups." Mashups [3] can be supported at both the client and the server sides, but in either case the focus is consistently on data aggregation. In contrast, SOA composition focuses on behavioral aggregation of services. This paper presents an approach to deliver composition capabilities in a resource-centric environment, such that data and behavioral compositions are seamlessly supported by a common workflow oriented model.

The approach taken is to adapt well known workflow techniques to the resource-centric model, and to extend it beyond simple resource interactions

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 94–106, 2007.

to cover fully asynchronous, interactive processes. However, since our goal is to deliver native Web workflow composition, matching the interaction and modeling principles of the Web is not enough. This research also paid special attention to lowering the development overhead of existing workflow models in order to address the short-cycle, highly iterative development model prevalent in the Web.

The result of this work is "Bite," a minimalist choreography language and runtime built to support the Web. Bite offers a workflow based development model for server-side scripting of all kinds of applications that interact with browser clients, e-mail clients, REST resources, remote functions available through URL encoded RPC, JSON-RPC, and local functions available through Java or JavaScript method invocations. Bite supports low overhead development by enabling a script oriented approach in which developers can choose what advanced capabilities to use according to the problem requirements. Variable and interface typing are not required, but are supported. Likewise, simple data flows can be created with the use of just a few constructs of the language, which is also able to support powerful long-running asynchronous processes including conditional and parallel processing.

A significant base of internet applications accessible through HTTP interfaces is currently available from Web sites such as Google, Yahoo, EBay, PayPal Amazon and many others, demonstrating a significant body of practice and commerce built around straightforward Web protocols. Bite provides a simple to use, solid composition model to leverage this growing trend.

The rest of this paper is organized as follows. Section 2 reviews prior work in the area. Section 3 presents an overview of the Bite language and its design principles. Section 4 explains how the Bite model addresses two major forms of Web composition, data and interactive flows. In Section 5 a sample Bite process is discussed in detail, and in Section 6 we discuss the implementation of the Bite runtime. Finally we present the conclusions of this work and new research directions in Section 7.

2 Related Work

The most relevant source of related work refers to Web-based workflows. We use the BPEL language [4] as our reference for service oriented process models. For a full survey of other approaches in Web services composition, see [5]. In this section, we focus on workflows that operate using the Web in a first class manner. Prior research can be summarized in the five categories below.

- **State machine based workflow.** A finite state machine is used in [6] to provide REST-centric, workflows that interact with a browser. The goal is to support single browser applications in which clicking on a link or posting a form results in the state machine transitioning to a new state.
- **Continuations.** A continuation [7] is a low-level programming primitive that stores execution context at a pre-determined location in the code, allowing different mechanisms to restore it later. A continuation point is associated with a wait state in the "flow" and with an event (such as an

incoming HTTP request) that will trigger restoration of context and allow execution to continue. Continuations are available in several languages such as Ruby[8] and Scheme, and externally supported for others such as Cocoon's FlowScript API [9] or JavaFlow[10]. Continuations support "flow-like" programming in traditional Web programming languages. One can send a user a form that contains a unique identifier of the continuation while maintaining a "continuations repository" [11,12]. Once the user fills out the form, the application knows exactly which continuation to go to. Anton van Straaten [13] advocates making the continuation itself a REST resource, giving each a URI.

- **Web Services Derivations.** In [14], the authors introduce a BPEL-like workflow for browser interaction in a REST-centric manner. Factories and process entry points are associated with externally visible URIs, and specialized semantics are provided for certain HTTP operations. It provides a single client model. Other proposed workflow models that use Web-centric interactions, but extend the HTTP verbs with additional commands, include SWAP, ASAP and Wf-XML.
- **Meta-data driven.** Another approach is to overlay meta-data on top of a service's implementation, such that the metadata describes the workflow semantics and directs the interaction with a browser. The Web Calculus [15] defines a directed graph where the nodes are document nodes and the edges may have closures. A client interacts with the service described by such a graph using a combination of graph-traversal and closure invocations.
- **Data Flows.** Examples of pure data flow approaches include Yahoo Pipes [16] and XProc [17]. However, they focus on manipulating data in response to a single incoming request. They are not geared to aggregating user interactions.

Bite shares certain aspects of its interaction model with [14], but extends an array of capabilities that make it particularly well adapted to the Web interaction model (including multi-protocol support) and different types of workflows (multi-party asynchronous flows and also data flows). The next Section describes the Bite approach in detail.

3 The Design of a Web-Centric Flow Language

Designing a process language for a REST oriented environment like the Web requires adapting the two-level programming model underlying workflow development to the resource-centric view. In addition, any programming model for Web applications needs to support the short-cycle, highly iterative development practice enabled by such systems as PHP and Ruby. In this paper we investigate the adaptation of BPEL's composition model to satisfy these two requirements. The goal is to leverage the accumulated experience of process-centric composition in SOA environments to deliver process composition in a Web environment.

We consequently need to address two major concerns: how a process executes within a REST environment, and how to support the Web's fast paced,

lightweight development model. Before explaining how this is done, we present a brief overview of the Bite language.

3.1 Bite Language Summary

As with most workflow languages, Bite contains two main constructs: activities and links. Activities define units of work and links define dependencies between activities. As in BPEL, activities have a "joinCondition" based on the status of the incoming links and links have a "transitionCondition." The execution semantics of links and activities is the same as <flow> in BPEL with "suppressJoinFailure" set to "yes," which itself is derived from FDL [18].

The language comes with a predefined set of basic constructs, shown in Table 1. The small set of built–in activities was chosen to embody basic actions in Web workflow, as described in the Notes column. However, additional activity types can easily be added by the user/developer community: the activity set is extensible as explained in section 3.2. The rest of this paper will elaborate on the different aspects of these constructs, with examples in Section 5.

Table 1. Overview of Bite Constructs

Activities	**Notes**
<receive>, <reply>,<receive-reply>	Receiving and replying to messages. Optional relative url attribute may be used to match incoming message. <receive-reply> shorthand for the two activities linked together, for the common pattern of callers just retrieving data
<invoke>	Call to an external party. Mandatory "invocationTarget" attribute, whose value is an expression, inlines service location and must resolve to a URI. Optional content-type and httpMethod attributes.
<local>	Call local code, such a static Java method or a script.
<wait>, <empty>, <terminate>	Utility activities: wait for fixed time, no–op, terminate the process instance.
<assign>	Basic data manipulation.
<pick>	External choice: contains an ordered list of external request and/or timer "choice" elements.
<while>	Loop as long as a condition is true.
Other Constructs	**Notes**
<source>	Control link. Also behaves as a data link if the "input" attribute is set to "yes."
<variable>	Optional variable declaration. May contain a "content-type" attribute, among others.

3.2 Deep Integration with the Web

Processes as active resources. In a SOA-centric model, a deployed business process interacts with its environment by invoking external services and by offering itself to requesters as a service over one or more service endpoints [4]. Likewise, in a REST oriented environment a process should interact with other entities as resources, and be itself exposed as a resource.

There is a deep similarity between the BPEL implicit factory model (in which a startable receive generates a new process instance for an incoming message), and the ATOM protocol by which a POST request creates items in an ATOM

collection [19]. We thus model a deployed process as a logical collection whose members are process instances. The process itself is exposed as a collection resource whose URL address corresponds to the startable receive of the process (see [4]). An HTTP POST against the process URL results in the creation of a new logical "item"– a process instance in the process collection. Following [19], a new URL is assigned to the newly created instance (resource), and returned in the HTTP Location header.

REST interactions on the new process instance URL have a specific meaning, providing process management calls not available to regular clients. GET and DELETE verbs respectively retrieve a representation of the process's state and terminate the running instance. A PUT request is not defined in Bite. Bite process instances are "active" resources with lifecycle and termination controlled by the internal logic of the process execution.

To support interaction between external requesters, a process instance exposes one or more URLs as logical addresses of the instance's nested resources. POST requests directed to these URLs are dispatched to the individual <receive> activities in the process model using the relative URLs defined in the activities' url attribute.

In BPEL, the partner link construct represents external partners (applications or people, see [20]). Bite represents external partners using their resource identifiers. Requests initiated by the process create HTTP requests (usually but not strictly GET or POST) directed at one of these external resources.

One note of caution is in order. The operation of the Web relies on more than REST interactions. Other protocols, in particular e-mail exchanges, are fundamental components of most complex Web interactions. For that reason, any workflow language directed at Web applications must be able to support alternative interaction protocols, and e-mail in particular. Bite's <invoke> activity, described in Section 3.1, enables processes to send generic invocations in different protocols identified by the scheme of the invocation target URI: "mailto:" sends an e-mail over SMTP and "http:" sends an HTTP request.

Dynamic data types. One characteristic of web interactions is the runtime discovery of request metadata, of which content-type [21] is particularly important. HTTP requests and responses carry content-type information used by the requestor's application to interpret the response. Bite supports dynamic content-type for incoming messages as well as optional statically defined content-type for outgoing requests (<invoke>).

Bite variables are associated with a content-type. The content-type of a variable used to save an incoming request is automatically set to the content type of the incoming message. As the variable gets used by the process, the content type is carried with the data. It is set in the corresponding HTTP header when the data is sent out. The result is that a flow may be designed to operate with different incoming content-types (such as XML and JSON), as long as no dependency on the specific data format is built into the code.

Content type can be statically set in the process definition for both variables (if the variable is declared) and <invoke> activities. An error is generated when

a content-type mismatch is detected between data copied into a variable and a static content-type declaration.

Extensible activity set. Bite's activity set is extensible, enabling communities of users to define domain specific activities in addition to Bite's built–in ones. Such new activities may capture well known actions such as data sort, append, etc. Bite provides for extensions by a tag library model similar to that of Java Server Pages ([22]). To define a new activity, developers register an XML parsing class and associate it with an execution class in the tag library registry. The parsing class will read the information provided in the activity definition within a process model and make it available to the execution class. At runtime, the Bite engine invokes the execution class providing access to the activity definition. The execution class gets its input from the process and writes its output back to the process; it is not given read or write access to any other part of the process state.

3.3 Lightweight Process Model

The workflow development model provides significant advantages over traditional procedural and object oriented implementation languages. Foremost is its ability to capture the end-to-end business logic of an application in a single definition. With long-running, asynchronous flows, traditional development models (such as servlets) necessarily fracture the application logic into multiple separate code artifacts. The result is the obfuscation of the coordination mechanisms by programming constructs such as hash tables, state machines, etc.

In order to deliver this value to Web developers it is crucial to offer a radically simplified workflow model and development process. In this section we examine how these two aspects are addressed by the Bite language.

Flat graph model. Much of the barrier to entry for BPEL is in its combination of flat and structured programming models. Bite's process model is a graph model with no nesting (except for loops), but with rich execution semantics similar to BPEL activities within a BPEL flow activity. Because of the lack of scope nesting, exception handling in Bite is fundamentally different from BPEL's [23]. Exceptions may be handled at the activity level through exception-labeled outgoing links as in [18]. Otherwise, they may be handled at the process level with an exception handler block.

Two of BPEL's structured activities find their way into Bite: <while> and <pick>. Structured iteration loops (<while>) significantly simplify the definition of correct iterative flows (as opposed to unstructured loops built using backward links). An example is shown in section 5.

The <pick> activity allows the flow to react to an exclusive choice from a set of different possible external inputs. External choice is a required feature [24] of interactive processes. Bite adapts the pick construct to the "flat graph" model by turning it into a flat activity whose output variable contains: which choice was taken (using an index or the choice's name if provided), and the received message data. The process may use the variable like any other, especially in link transition conditions to go down a different branch based on the selected choice.

Workflow scripting. Most of today's workflow languages are strongly typed with respect to both data and behavior (interfaces). There is a clear rationale for strongly typed languages in general (ability to detect errors, overall consistency, etc.) and workflows in particular. From the practice of Web application development, however, we have learned that the overhead imposed by typing and other forms of required artifacts external to the workflow logic itself creates a barrier of entry that excludes most Web developers. (See [2] for a good discussion on the topic.)

With this consideration in mind, Bite takes a "scripting" approach to workflow definition. By this we specifically mean:

1. The principle of "use implies definition:" Variables can be directly used without requiring prior declaration or explicit typing. This is similar to the use of variable in languages like JavaScript. However, a developer may choose to explicitly define and type a variable using the optional <variable> element.
2. The principle of "convention over configuration." Bite conventions dictate that the output of an activity is contained within an implicitly defined variable with the same name as the activity. Additionally, a control link may also specify that the output data of its source activity be used as part of the input of its target activity.
3. Radical reduction of extraneous constructs while eliminating levels of indirection. Invocation targets on invoke activities are encoded as literal URLs or as data variables (see section 5) No typing of the resource being accessed is required (i.e.: message types). Contrast this with the BPEL model, where an invocation must reference a partner link construct that is in turn typed by a predefined partner link type, which in turn depends on WSDL port types and XML Schema definitions, and which is finally bound to physical service endpoint by an implementation dependent mechanism that is out of the scope of the BPEL language.

Flexible configurability. Bite processes provide configurability by enabling values of variables to be set outside of the workflow definition. This is similar to "properties" in Java. This capability may be used for actions such as: late binding of partner URLs, or turning paths of a process on or off by setting values of variables used on transition conditions or the condition of a while loop.

4 Web Workflow Scenarios

We focus on two scenarios for which Web workflow provides significant value added— data-centric flows and interactive flows.

4.1 Web Data Flows

The resource-centric nature of many Web applications makes data integration common for simple Web integration scenarios. The approach is well illustrated by

the Yahoo Pipes tool [16], and its model is also captured by the XProc language [17]. The common pattern is a set of processing steps connected by explicitly stated data dependencies. Execution of a step takes place as soon as all required inputs are available.

This model is natively supported in Bite, taking advantage of the fact that a data dependency always implies a (direct or indirect) control dependency. The execution semantics of Bite imply that an activity targeted by a link, defined using <source> in Bite, waits for completion of the link's source before proceeding. An activity may contain <input value="..."> subelements that explicitly provide it with data. The value is an expression that of course may refer to any of the process's variables. The <source> element provides an "input" attribute that enables one to treat it as a combined control link and <input>. The source's "name" attribute refers to the link's source activity. If "input" is set to "yes," it indicates that the output of the source activity (contained in an implicit variable with the same name, see Section 3.3) is treated as one of the inputs of the target activity. Therefore, an activity's input data set consists of the ordered list of <source input="yes"> and <input> elements. The following code snippet shows a data flow connection between two activities, as do lines 2-9 of the example in section 5.

```
<invoke name="getBBCTopStories"
   invocationTarget="'http://rss.news.yahoo.com/rss/topstories'"/>
<local name="sort" invocationTarget="'java:util.Sort'" operation="sort">
   <source name="getBBCTopStories" input="yes"/>
</local>
```

Data flow composition is thus a particularly simple application of Bite's general workflow model. Data flows are typically executed synchronously in response to a single external request for data retrieval (such as through a GET request), and they have very limited error handling capabilities (see [17]).

The main value of encoding a data flow as a workflow lies in explicitly exposing data dependencies. Bite's support for more complete workflow execution semantics (including error handling and asynchronous execution) allows seamless extension of data flow logic into more functional workflows.

4.2 Interactive Flows

Most Web applications are highly interactive. Beyond delivering information to end users, they often receive customer data through HTML forms and contact customers back via e-mail. They often involve several parties and potentially back–end applications. Many typical Web transactions are potentially long running (resolved in the course of days or weeks) and asynchronous, involving a combination of synchronous HTTP interactions and asynchronous e-mail messages.

Bite's model is particularly well suited to support these types of applications. Remaining fundamentally Web centered, both in protocols and interactions models, it has significant advantages over other development approaches:(1) the application logic is defined in a single file where the interaction with all the parties

and their relationships are explicitly encoded; (2) the workflow model natively supports asynchronous execution, as opposed to object or procedural models; (3) multi-protocol capabilities support seamless integration of traditional Web interaction models, e-mail, and back-end interactions. In addition, Bite supports multi-party interactions natively since it supports a Web-centric version of BPEL's partner link model.

5 Example Bite Workflow: Special Order

The following example illustrates Bite's salient features. It demonstrates a mix of automatic and human interaction in a scenario involving multiple parties and agents. A customer requests a special order item at a high–end store as shown in Figure 1. The employee submits the order to the process (order) and gets back a URL of where to go to confirm receipt once the order arrives in the store. The process sends the order to an automated authorization service (autoApprove). If it is not approved, it goes to a manager via e-mail (rqstApproval) for a deeper evaluation. The manager gets a link in the e–mail notifying her of the order and approves or rejects the order (authorize). If the manager does not approve, the process ends. If either the service or the manager had approved the order, the process sends an e-mail to the designer to create the item (makeItem). Then, a loop is entered that waits until the employee confirms receipt of the item. In the pick (pick1), the employee has 7 days (reminder) to confirm (confirmation) after which he gets an e–mail reminder (remind) to find out why the item is delayed. If he confirms, he gets a reply acknowledging that (confirm). Once the process is notified that the item is in store, the customer is notified via e-mail (itemArrived).

We now look at the complete Bite process, shown below, and containing nearly all the language elements. Activity names match the labels in the figure, so we focus on highlighting interesting aspects of the script. Consider the receive-reply (lines 2–6). It receives the order from the client, at which time a process instance is created and the value of ProcessId is set, and replies with the value inlined in "input." ProcessId is a reserved variable available to every Bite process instance containing the id of that instance. The full URL is also available in the reserved Location variable. The reply contains a URL that is routable back exactly to this process instance: notice the "ProcessId."

The activity "autoApprove" (line 7–9) shows an example of a service invocation as well as a control link that transmits data. The "source" element has "input" set to yes, meaning that the message sent to the service is the message received from "order." The next invoke, "rqstApproval" (line 10–16) shows an e–mail style invoke. Notice the "mailto" scheme in the URL. The "invocationTarget" attribute takes an expression so one can build the value directly from the received message in "order." Recall that the default output variable of an activity has the same name as the activity. Therefore, the order, containing the manager's e–mail, is in the variable "order." Notice the URL used for the manager to send back a response: it will be received by "authorize" (line 17–20).

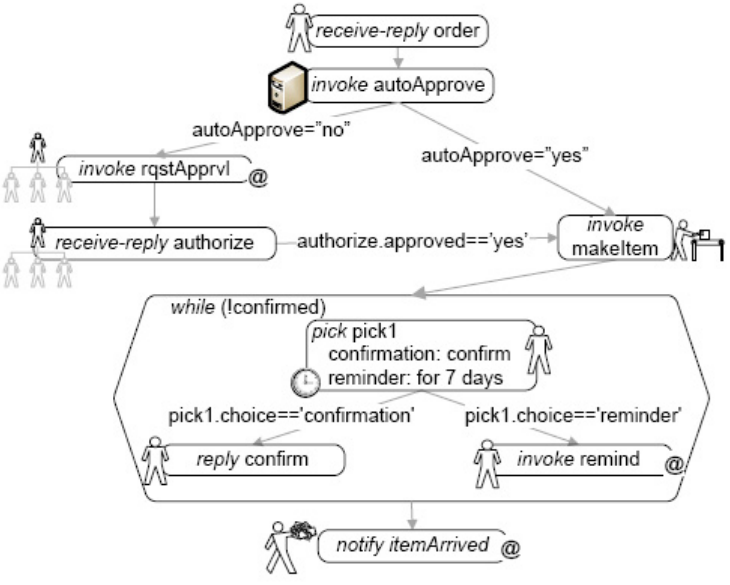

Fig. 1. Sample Bite process for a special order. Icons by activities represent the person or service the activity interacts with.

From here, an interesting part is the pick activity (line 29–32). It has a message–based choice (line 30) that waits for the employee to confirm an alarm (line 31). Notice how one uses the selected choice in the transition condition of the links entering "confirm" (line 34) and "remind" (line 39).

```
1.  <process name="orderItemPlus">
2.    <receive-reply name="order" url="/initiateCase">
3.      <input value=
4.       "'When the item arrives, confirm here: http://localhost:8080/demo/order/'
5.        + ProcessId + '/confirm'"/>
6.    </receive-reply>
7.    <invoke name="autoApprove" invocationTarget="'http://example.com/orderAuthorization">
8.      <source name="order" input="yes"/>
9.    </invoke>
10.   <invoke name="rqstApproval" invocationTarget=
11.          "'mailto:'+ order.managerEmail[0]" operation="Manager Approval">
12.     <source name="autoApprove" condition="autoApprove=='no'"/>
13.     <input value=
14.     "'Please go here to approve an order: http://localhost:8080/demo/approvalform/'
15.       + ProcessId"/>
16.   </invoke>
17.   <receive-reply name="authorize" url="/approvalResponse">
18.     <input value="'Thank you for responding.'"/>
19.     <source name="rqstApproval"/>
20.    </receive-reply>
21.   <invoke name="makeItem"  invocationTarget="'mailto:' + order.designerEmail[0]"
22.                            operation="Manufacturer Request">
23.     <source name="authorize" condition="authorize.approved[0]=='yes'"/>
24.     <source name="autoApprove" condition="autoApprove=='yes'"/>
25.     <input value="order"/>
26.   </invoke>
27.   <while name="loop" condition="!confirmed">
28.     <source name="makeItem"/>
```

```
29.     <pick name="pick1">
30.       <choice name="confirmation" url="/confirm" outputVariable="confirmed"/>
31.       <choice name="reminder" for="'P7D'"/>
32.     </pick>
33.     <reply name="confirm" url="/confirm">
34.       <source name="pick1" condition="pick1.choice=='confirmation'"/>
35.       <input value="'Thank you for confirming that this order has arrived.'"/>
36.     </reply>
37.     <invoke name="remind" invocationTarget="'mailto:' + order.employeeEmail[0]"
38.             operation="Employee Reminder">
39.       <source name="pick1" condition="pick1.choice=='reminder'"/>
40.       <input value="order"/>
41.     </invoke>
42.   </while>
43.   <invoke name="itemArrived" invocationTarget="'mailto:' + order.customerEmail[0]"
44.           operation="Customer Notification">
45.     <input value="'Your order is ready for pickup at the store.'"/>
46.     <source name="loop"/>
47.   </invoke>
48.  </process>
```

6 Implementation

The Bite language has been implemented as a set of embeddable Java components. A "BiteManager" (referred to simply as manager) implements the language's core execution semantics. A servlet is used to service incoming HTTP requests, forwarding them to the manager. This servlet has been tested through deployment into Jetty installations.

We now briefly describe the runtime operation of the Bite engine. Incoming requests to URLs matching the <receive> activities in the process model are mapped to execution events. The target process instance is identified from the request's URL, and its instance data is retrieved from a map of process context data. The manager uses a thread pool to serve requests to multiple concurrent process instances. The hand-over of events between worker threads in the thread pool and the servlet thread associated to the incoming request is supported using event queues stored as part of the instance data. In addition to <receive>, wait, pick, and invoke also wait for events coming from the manager notifying them, respectively, of when the alarm has gone off, a message or alarm matching a 'choice' has occured, or the response to the invocation has arrived. A worker thread navigates a process instance until all paths block, or the process completes. A path blocks when a receive, invoke, pick, or wait activity is encountered and there is no suitable event queued that can matches the activity. Such an activity is then added to a list of waiting activities for new events.

7 Conclusion and Future Work

This paper has presented the Bite Web–centric flow composition model through a discussion of its main design points and an overview of the language and implementation. Devlivering an composition mechanism for the development of Web applications and leveraging their workflow model, Bite supports explicit encoding of compositional logic in a single programming artifact.

The Bite model is aligned with the resource-centric view of the Web, but is not limited to REST interactions alone, in line with current practice in the Web. Bite supports sophisticated asynchronous, multi-part workflows as well as simple data composition ones. In Bite, workflows are developed with minimal up-front overhead, aligning the development model with fast paced development practices of Web scripting languages. The runtime has been developed in Java and tested on the Jetty servlet engine, but is designed as an embeddable component that can be used in other runtimes.

To fully exploit the potential of Web–centric compositions, we are starting new work in several areas. We are exploring a dedicated scripting syntax, as an alternative to Bite's current use of XML. While XML ensures wide familiarity among developers, a scripting alternative can improve readability and usability. We are also planning to identify and support well known interaction patterns, such as the e-mail and form interaction shown in Section 5, using Bite's tag library mechanism. We are considering a <choice-reply> under <pick> to mirror the <receive-reply> shortcut, and investigating whether a simplified form of BPEL correlation would be a useful addition. The difficulty there is in simplifying the definition of a correlation set especially for the case of untyped messages. Finally we are extending our effort to provide increased transparency and control with respect to the use of interaction protocols by exposing HTTP and e-mail artifacts such as protocol headers directly in the flow language.

Since the submission for publication, a later version of the Bite language and runtime, under the title "the Project Zero assembly flow language," has become publicly available [25].

Acknowledgement. The authors would like to thank Marc-Thomas Schmidt for his comments and advice regarding the general design of the Bite language and Xin Sheng Mao for his input on the use of Bite for data flows.

References

1. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice-Hall, Englewood Cliffs (2005)
2. Bosworth, A.: ICSOC 2004 keynote talk. Adam Bosworth's Weblog (2004), http://www.adambosworth.net/archives/000031.html
3. Anonymous: ProgrammableWeb.com (2007), http://www.programmableweb.com/
4. OASIS: Web Services Business Process Execution Language Version 2.0. (2007), http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
5. Dustdar, S., Schreiner, W.: A survey on web services composition. Int. J. Web and Grid Services 1(1) (2005)
6. Kuhlman, D.: Workflow and REST how-to. Personal Web site (2003), http://www.rexx.com/~dkuhlman/workflow_howto.html
7. Ruby, S.: Continuations-for-curmudgeons. Blog post (2005), http://www.intertwingly.net/blog/2005/04/13/Continuations-for-Curmudgeons

8. Thomas, D., Fowler, C., Hunt, A.: Programming Ruby: The Pragmatic Guide, 2nd edn. Addison-Wesley, Reading (2004)
9. Apache: Apache Cocoon, Control Flow. (2006), http://cocoon.apache.org/2.1/userdocs/flow/index.html
10. Apache Jakarta: Javaflow (2006), http://jakarta.apache.org/commons/sandbox/javaflow
11. Tate, B.: Crossing borders: Continuations, web development, and java programming (2006), http://www-128.ibm.com/developerworks/java/library/j-cb03216/?ca=dgr-jw22StatelessWeb
12. Belapurkar, A.: Use continuations to develop complex web applications. IBM developerWorks (2004), http://www-128.ibm.com/developerworks/library/j-contin.html
13. Straaten, A.V.: Continuations continued: the REST of the computation (2006), http://ll4.csail.mit.edu/slides/rest-slides.pdf
14. zur Muehlen, M., Nickerson, J.V., Swenson, K.D.: Developing web services choreography standards - the case of REST vs. SOAP. Decision Support Systems 37 (2004)
15. Waterken Inc.: Web-Calculus. (2005), http://www.waterken.com/dev/Web/Calculus/
16. Yahoo Inc.: Yahoo pipes (2007), http://pipes.yahoo.com
17. Walsh, N., Milowski, A.: XProc: An XML pipeline language. Working draft, W3C (2007), http://www.w3.org/TR/xproc/
18. Leymann, F., Roller, D.: Production Workflow. Prentice Hall, New York (2000)
19. Gregorio, J., de hOra, B.: The atom publishing protocol. Internet draft, IETF Network Working Group (2007), http://bitworking.org/projects/atom/draft-ietf-atompub-protocol-15.html
20. Active Endpoints, Adobe, BEA, IBM, Oracle, SAP AG: WS-BPEL extension for people (BPEL4People). IBM developerWorks (2007), http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/
21. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Mastinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol – http/1.1. Request for Comments 2616, IETF Network Working Group (1999), http://www.ietf.org/rfc/rfc2616.txt
22. Sun Microsystems: JSR-000245 JavaServer Pages^{TM} 2.1. (2004), http://jcp.org/aboutJava/ communityprocess/final/jsr245/index.html
23. Curbera, F., Khalaf, R., Leymann, F., Weerawarana, S.: Exception handling in the BPEL4WS language. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, Springer, Heidelberg (2003)
24. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press, Cambridge (1999)
25. IBM: Project zero (2007), http://www.projectzero.org/