Business process enactment

Chapter 8
Business Process Enactment

Georgios Kouvas, Paul Grefen, and Ana Juan

In this chapter, we discuss the business process enactment environment of the CrossWork architecture, providing support for network business process enactment and legacy system integration. We first discuss the overall approach to enactment, which consists of global enactment and local enactment, supported by two modules as described in the architecture design (see Chapter 5). Details of the Global Enactment module are discussed in Section 8.2. The discussion of the Local Enactment module follows in Section 8.3. As described in the architecture, enactment is coupled to Legacy Integration. In Section 8.4, we therefore discuss the Legacy Integration module. We conclude this chapter in Section 8.5.

8.1 Overall Approach to Enactment

After a Business Network Process has been composed and verified, it is ready for enactment by an IVE. Enactment of a Business Network Process is based on a two-level mechanism, consisting of global process orchestration and local process execution. Both levels are designed such that they allow flexible enactment topologies through the use of remote workflow clients [6].

To assure interoperability with industry-standard process enactment platforms, the industry-standard Business Process Execution Language (BPEL) [2] is used for enactment. BPEL is an XML-based language, built on top of SOA and Web Services specifications. It is used to define and manage long-lived service orchestrations or processes. In BPEL, a business process is a large-grained Web Service, which executes a control flow to complete a business goal. The steps in this control flow execute activities that are centred on invoking partner services to perform tasks and return results to the process. The drivers for choosing BPEL in our approach are manifold. Firstly, enterprises are evolving their SOA implementations from simple, fine-grained services, to more complex, large-grained services. Secondly, enterprises are employing service-oriented architecture strategies for integration.

G. Kouvas (✉)
Exodus, Athens, Greece
e-mail: gkou@exodus.gr

N. Mehandjiev, P. Grefen (eds.), *Dynamic Business Process Formation for Instant Virtual Enterprises*, Advanced Information and Knowledge Processing,
DOI 10.1007/978-1-84882-691-5_8,

Thirdly, in response to the first two items, vendors are creating integration and SOA infrastructure solutions that offer BPEL orchestration, and/or use BPEL for internal processing. Finally, the specification is maturing; BPEL 2.0, sponsored by the Organization for the Advancement of Structured Information Standards (OASIS), is now available.

Figure 8.1 depicts the CrossWork enactment architecture. The two levels of workflow enactment (global enactment and local enactment) are clearly identified in the enactment architecture. Next to this, we employ a monitoring user interface module.

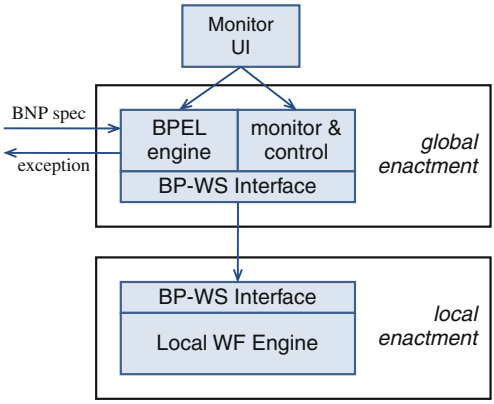

Fig. 8.1 Enactment architecture

The Global Workflow Specifications (shown as Business Network Process Spec in Fig. 8.1) expressed in BPEL are obtained by automated translation of eSML process specifications. For this reason, an eSML2BPEL translation submodule is included in the global architecture. This submodule represents the bridge between the IVE formation and the enactment phase. For the real-time enactment of the Global Workflow, a standard BPEL engine is used. It reads BPEL process definitions (and other inputs such as WSDL files) and creates representations of BPEL processes. When an incoming message triggers a start activity, the engine creates a new process instance and starts it. The engine takes care of persistence, queues, alarms and many other execution details.

During the enactment of the global process and depending on the control flow, local partner Web Services (representing local business processes) are invoked either asynchronously or synchronously to perform their job. A precondition is that local partners have already exposed their business processes as Web Services and they have already exposed a WSDL interface. But this is not enough. In dynamic Business Network Process management, a Web Service that represents a business process, in contrast with a traditional Web Service, is often required to offer external visibility onto its internal process structure. For this purpose, we use the concept of BP-WS, introduced by [7], which includes a business process specification and business process state that can be accessed externally. Access to specification and state is provided through a number of dedicated Web Service interfaces. Grefen

et al. [7] introduces four BP-WS classes following four control flow interface levels, which we use as a basis here. According to its internal business logic (or other initiatives) each local partner may decide to expose (or hide) details of its services by using the BP-WS classes. Local partners specify selected internals of their business processes (a projection of the business process specification) to the outside world by using eSML (for workflow composition) and BPEL (for monitoring and control purposes).

The concept of developing business process specifications (in BPEL) for the local Web Services enables the overall monitoring of the enactment at both local and global levels. For that, stateful global BPEL specification files need to be combined with stateful local specification files. In our approach, we expand the capabilities of the global Enactment Engine by adding monitoring and control functionalities; a Monitoring and Control engine. The purpose of this engine is twofold. Firstly, it is used to communicate with all specification and control ports of all local and global services in order to be aware of the combined status of execution. Secondly, it distributes all control messages to the appropriate local services through the control ports.

8.2 Global Workflow Enactment

In the previous chapter, we have seen how a global business process is composed and verified. This global business process is next enacted by the Global Enactment module, which interfaces to the Local Enactment modules, i.e. the local workflow management environments in each of the IVE member organizations. We first discuss the BPEL language used for the global business process representation and describe its process and activity structure. Then, we describe the BPEL engine selected for the actual implementation of the Global Enactment module. After having discussed these ingredients, we describe the architecture and realization of the Global Enactment module in detail.

8.2.1 BPEL Language

In the enactment phase, the language that we use for the representation of a Global Business Process is BPEL. While BPEL is not new – Version 1.1 of the specification was finalized in 2003 – enterprise uptake of BPEL is just recently gaining momentum. The specification is maturing; OASIS has announced the release of BPEL 2.0 as an accepted standard.

BPEL, also known as BPEL4WS and WSBPEL, is an XML-based language built on top of Web Service specifications, which is used to define and manage long-lived service orchestrations or processes. In BPEL [8], a business process is a large-grained stateful service, which executes steps to complete a business goal. That goal can be the completion of a business transaction, or fulfilling the job of

a service. The steps in the BPEL process execute activities (represented by BPEL language elements) to accomplish work. Those activities are centred on invoking partner services (e.g. Web Services specified in WSDL) to perform tasks (their job) and return results to the process. The aggregate work, the collaboration of all the services, is a service orchestration.

If BPEL is used for orchestration, the primary service is a BPEL process, because BPEL processes are implemented as services. The BPEL process controls the overall sequence and invokes the collaborating services. To perform the orchestration, the BPEL process contains the logic (sequence, activities, invocations, assignments and case logic) to invoke other services (collaborators) to complete their combined job. The BPEL process may be asynchronous or synchronous. In an asynchronous model, the BPEL process is managing a long-lived flow of work; in other words, a process. In a synchronous mode, the BPEL process is more likely playing the (logical) role of another service type, such as request/reply, worker, agent or monitor.

BPEL's technology underpinnings are XML and Web Services. XML, as commonly known, is a standard, tag-based markup language used in document and message definition and processing. In addition, XML serves as the language base (format and syntax) for many special purpose languages such as WSDL (Web Services Description Language), and RSS (Really Simple Syndication).

BPEL follows this same model. The BPEL process description is encoded using XML language constructs. BPEL also incorporates XPATH to write expressions and queries. Web Services are important to BPEL in two ways. First, BPEL processes follow the WSDL service model, which is why BPEL processes are implemented as services. More specifically, BPEL processes have a WSDL definition, and are implemented as Web Services (usually compliant with WSDL, SOAP, and UDDI). Along with the basic functions of service definition, discovery and invocation, the Web Service implementation allows BPEL execution engines to leverage additional Web Service standards at runtime, such as WS-Addressing and WS-Coordination. Second, the collaborating services are also Web Services, described in WSDL. So, BPEL processes and collaborators are Web Services, and BPEL execution engines contain (and use) Web Services runtime components. Natively, the collaborating services (and the process service) are Web Services. However, BPELJ, an extension of BPEL, allows for a mix of Java Services and Web Services.

BPEL offers a good model to separate orchestration logic from the participating services, and process configuration using BPEL provides many advantages over hard coding of service interactions. However, there is a processing overhead and infrastructure expense, so BPEL might not be the best choice for simple orchestrations. As a rule of thumb, a simple orchestration is comprised of two to five services and has static interaction patterns. As a language to develop processes, BPEL is good at executing a series of activities, which occur over time, and interact with internal and external services. These processes may represent IT scenarios, such as integration, or business scenarios, such as information exchange, or flows of work.

As for its limitations, BPEL does not account for humans in a process, and as a result does not provide a traditional workflow management, as there are no concepts for roles, tasks and inboxes. In addition, BPEL does not support very complex business processes, those which evolve during their execution, branching out to incorporate new parties and activities. Lastly, BPEL does not have native support for Business Activity Monitoring (BAM). There is no data model for measurement and monitoring.

The BPEL environment is divided into five major parts (see Fig. 8.2), "Designer", "Server", "Server Host Environment", "Management Console" and "Process Database".

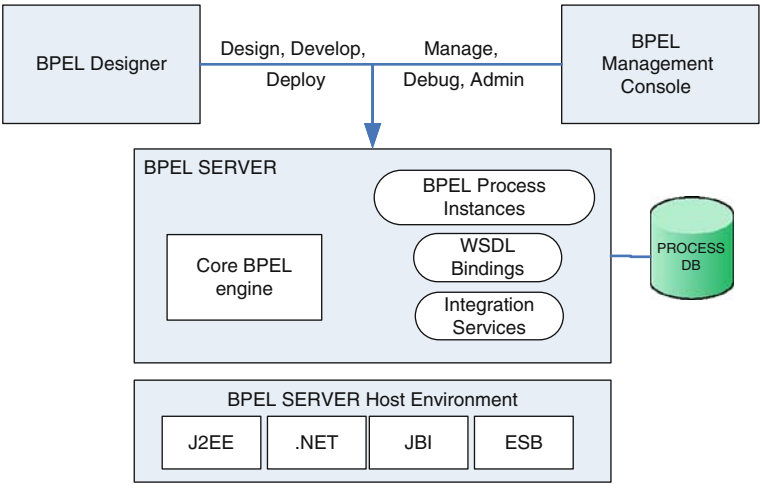

Fig. 8.2 Overview of BPEL processing

The "BPEL Designer" enables the user to design, develop and deploy BPEL processes. Here, we find the tools for business analysts and developers/integrators to describe processes (orchestrations) composed of steps (activities), associate services and add any special business or validation logic to the process flow. Additionally, partner links can be added and services located through the Universal Description, Discovery and Integration (UDDI) browser. We can also use function and copy wizards. These tools allow the user to perform these activities visually in a graphical environment without having to write BPEL code by hand. Most tools support a toggle between the graphical design view and the BPEL source code view. Changes made in one view should be reflected in the other. The Designer can deploy the developed processes directly to the BPEL Server. This eases the development and maintenance of BPEL processes considerably.

In the centre of the diagram is the "BPEL Server" (or BPEL execution environment). The "BPEL Server" contains the executing "BPEL process instances", the "WSDL bindings", the "integration services" and the "core BPEL execution

engine". The "core BPEL engine" is the runtime environment where the BPEL processes are deployed and executed. It provides support for the process life cycle requirements (instantiation, communication, dehydration/hydration, correlation, transaction management, compensation and termination) as described in the specification. The "WSDL binding" framework is responsible for communication with the BPEL processes deployed on the server. This includes clients that would like to access a BPEL process and BPEL processes that would like to access other Web Services (partner links). The "Integration services" enable performing transformations of XML documents that go beyond the support of XPath. This is very useful especially if business processes described in BPEL communicate with Web Services and exchange XML documents.

To the right of the BPEL server, we see the "Process Database" (DB). The "Process DB" is used to hold information on process instances and state. Above the DB, we see the BPEL server's Management Console. With that console we can deploy, manage, administer, and debug BPEL processes. The most important features of the BPEL Console include visual process flows, audit trails, debugging view of processes and process history. Finally, underneath the BPEL Server, we see four different host environment options; "J2EE" container, ".Net" container, "Java Business Integration" (JBI) container, and an "Enterprise Service Bus" (ESB). Example implementation for each host environment includes; Oracle's BPEL Process Manager, Microsoft's BizTalk Server, FiveSight Technologies' PXE, and CapeClear's ESB.

8.2.2 BPEL Processes and Activities

A BPEL process consists of the following components:

- *Partner links*: relationships between two Web Services at the interface level;
- *Partners*: entities taking part in a Web Service transaction;
- *Variables*: containers for values;
- *Correlation sets*: sets of data that uniquely identify the business process; at different times in the process, different correlation sets may identify the process;
- *Fault handlers*: describe what to do when problems occur;
- *Compensation handlers*: describe how to reverse already-completed business processes;
- *Event handlers*: handle incoming messages and alarms;
- *A top-level activity*: a single BPEL activity, usually a container for other activities.

Activities are the building blocks of BPEL processes. Basic activities exhibit conceptually simple behaviour like receiving a message, invoking a Web Service and assigning values to variables. Structured activities are similar to conditional and looping constructs in programming languages. Special activities introduce variable

scoping and handle abnormal activities such as process termination and compensation (explicitly undoing a process). Activities are joined by links, either explicit or implicit. The path taken through the activities and links is determined by many factors, including the values of variables and the evaluation of expressions. Every activity exists within a scope, which is a context for variables, fault handlers and compensation handlers. Scopes are conceptually similar to programming language blocks that introduce new variable scope and (depending upon the language and construct used) exception handling mechanisms. Some activities such as "scope" and "invoke" generate new scopes, whether implicitly or explicitly. Below, we list the activity types supported by BPEL.

Activity	Notes
Basic activities	
<receive>	Block and wait for a message from a partner
<reply>	Reply back to the partner that sent the message we received
<invoke>	Call some other Web Service, either one-way or request-response
<assign>	Assign or copy values to variables
<throw>	Generate a fault
<wait>	Wait for a given time period (time-out) or until a particular time has passed (alarm)
<empty>	A no-op
Structured activities	
<sequence>	Execute "child" elements in order
<switch>	Just like a "switch" or "case" statement
<while>	Repeat an activity while a condition is true
<pick>	Block and wait for a message, time-out, or alarm
<flow>	Children are executed concurrently; links can provide additional control structure
Special activities	
<scope>	Define a new scope for variables, fault handlers, and compensation handlers
<compensate>	Invoke compensation on an inner scope that has already completed normally
<terminate>	Immediately terminate a business process instance

Each activity has an associated state. The activities enter or exit these states based on the rules of BPEL. The activities also fire events to notify listeners of their changes in state. There are mechanisms in place for listening to these events. An activity must be in one of the following states (defined in AeBpelState):

8.2.3 BPEL Engine

A BPEL engine is a software module that reads BPEL process definitions (and other inputs such as WSDL files) and creates representations of BPEL processes. When an

State	Notes
INACTIVE	All BPEL activities are in the inactive state when the process starts
READY_TO_EXECUTE	Ready to execute. These activities have been queued by their parent and their join condition has evaluated to true
EXECUTING	Currently executing
FINISHED	Finished executing without a fault
FAULTED	Finished executing with a fault
DEAD_PATH	Removed from the execution path due to dead path elimination. When a parent activity's state becomes dead, that state is propagated to all of its children
QUEUED_BY_PARENT	Queued for execution by their parents
TERMINATED	Terminated
Unknown	The activity's state is null; when a parent activity's state becomes unknown, then the children's states change to INACTIVE

incoming message triggers a start activity, the engine creates a new process instance and starts its execution. The engine takes care of persistence, queues, alarms and many other execution details.

In CrossWork, we have used the ActiveBPEL open source software [1]. The ActiveBPEL engine is a commercial-grade open source implementation of the BPEL for Web Services Version 1.1 specification and is fully compliant with that specification. The ActiveBPEL engine is a robust runtime environment that is capable of executing process definitions created for the BPEL standard. The ActiveBPEL engine technology is developed and maintained by Active Endpoints, which also uses the same technology in its commercial products. Active Endpoints has created the ActiveBPEL open source project based on the belief that the open source model is an effective means through which to foster community interest, education and development around the BPEL standard. The ActiveBPEL engine runs in any standard servlet container such as Apache Tomcat.

Figure 8.3 shows the architecture of a BPEL engine. The "Database" elements on the right of this illustration represent generic persistent storage. The ActiveBPEL engine uses a pluggable architecture; different persistence managers may implement different storage mechanisms. The ActiveBPEL engine ships with a persistence manager that keeps everything in memory.

An engine factory manages the creation of an ActiveBPEL engine. Separation of responsibilities is achieved through the use of objects that handle services such as queue management, alarm and timer services and process deployment. Engine configuration is handled by an object that supplies default values and reads the aeEngineConfig.xml file. The engine is connected to a queue manager and a process state manager, and objects that are responsible for performing these services for the engine. The engine is highly configurable. aeEngineConfig.xml specifies not only values like cache sizes and logging state, but also determines the class of various factories, managers and handlers. A process deployment provider handles reading

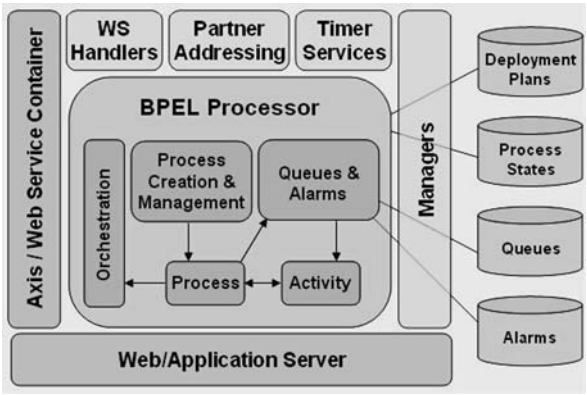

Fig. 8.3 BPEL engine architecture (adapted from [1])

process deployment descriptor files, and a process deployment manager handles process creation. A work manager schedules asynchronous operations.

Each BPEL process must have at least one start activity. A new BPEL process is created when one of its start activities is triggered, either by an incoming message or by a Pick activity's alarm. The engine dispatches incoming messages to the correct process instance. If there is correlation of data, the engine tries to find the correct instance that matches the correlated data. If there is no correlation of data and the request matches a start activity, a new process instance is created. When the engine reads a BPEL process definition, it creates objects called activity definitions that model the process. Activity definitions contain all of the information required to instantiate a BPEL activity implementation object. In this regard, definitions are analogous to classes while the implementation objects are analogous to instances of those classes (objects). Both the engine and its event listeners have access to these definitions. The events contain an XPath value that indicates which activity in the process is triggering the event. These XPath values come from the activity definitions.

The engine creates its implementation objects by using a "Visitor" pattern to visit the activity definition object model, creating implementation objects from this model, as presented in Fig. 8.4. The ActiveBPEL engine encapsulates any implementation logic within this construction process. For example, an implicit scope within an invoke activity will generate an explicit scope with a single invoke child

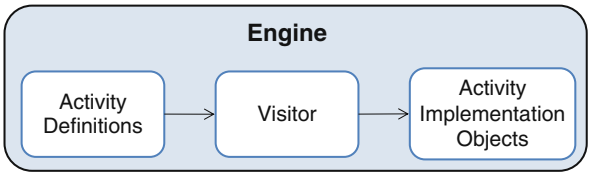

Fig. 8.4 The "visitor" pattern (adapted from [1])

activity. The designer or other listeners do not have to know about these implementation decisions since they are only aware of the definitions and their XPath information.

The ActiveBPEL engine itself does not handle input and output. Instead, protocol-specific handlers such as AeBpelRPCHandler and AeBpelDocumentHandler translate data from a particular protocol to a message and vice versa. All variables implement the IAeVariable interface. This interface has the ability to get the definition and the payload, which is different if the variable is declared as a type versus an element or message. The message payload requires an interface for detailed interactions with the part objects.

All activities and links allow expressions for various attributes of the object. These expressions require a consistent method for execution and a description of the execution context. The BPEL object is itself a scope (in the variable sense of the word) and can be used to correctly retrieve the accessible variables of the expression context. Evaluation allows all the XPath extensions which have been documented in the BPEL specification (for example, bpws:getVariableData). During process execution, the ActiveBPEL engine fires events indicating its progress. When logging is turned on, an instance of AEEngineLogger listens for engine events and writes them to individual process log files. Once the process completes, the file will be closed.

Each BPEL process must have at least one start activity. A new BPEL process gets created when one of its start activities is triggered, either by an incoming message or by a Pick activity's alarm. The engine dispatches incoming messages to the correct process instance. If there is correlation between data, the engine tries to find the correct instance that matches the correlated data. If there is no correlation of data and the request matches a start activity, a new process instance is created.

The "receive queue" contains the currently executing receive activities across all process instances. Receive activities include onMessage activities that are part of a "pick" or an event handler. A "receive activity" is said to be executing when it has been queued by its parent (for example, a scope, flow or sequence) but has not yet received the message from the outside world that it is waiting for. The "receive queue" also contains inbound messages from the outside world that did not match a waiting "receive activity" already in the queue and were themselves not capable of creating a "new process" instances. An unmatched receipt of data like this is possible given the asynchronous nature of some Web Services. The engine will accept these unmatched messages provided that they contain correlation of data. These messages are queued until a time-out period passes. The period is specified by the engine configuration parameter *UnmatchedReceiveTimeout*. When a process queues an activity like a *receive*, then this activity will stay in there until the data arrives or the process terminates (through a fault or terminate activity). Picks are slightly different; the first onMessage or onAlarm to match for the pick immediately sets the state of all of the other possible messages/alarms for that Pick to DEAD_PATH. This will remove them from the queue. Event handlers automatically remove their queue entries once the scope that defines them completes.

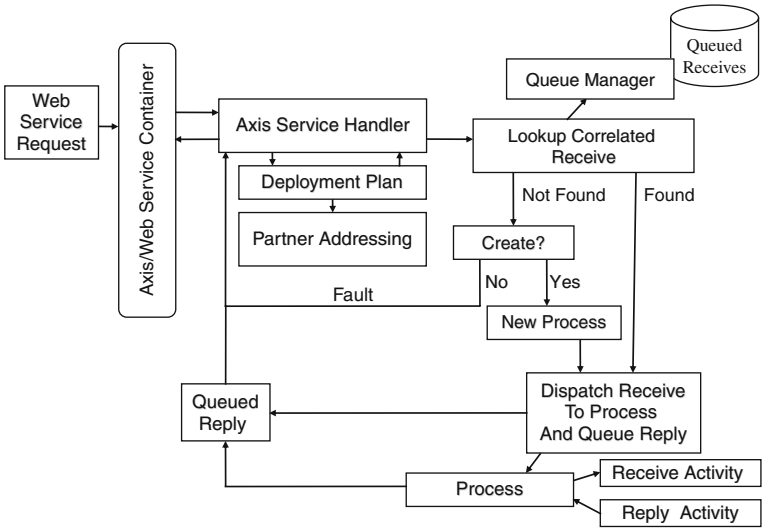

Fig. 8.5 BPEL request handling (adapted from [1])

8.2.4 Global Enactment Subsystem

The Global Enactment module is a runtime environment that is capable of executing process definitions created by the Workflow Composition module and expressed in BPEL 1.1 specifications. Moreover, the Global Enactment module is responsible for gathering and storing all monitoring information created during the execution of the process definitions. In more detail, this includes the following functions:

- Deploying of a process definition (for execution);
- Executing an instance of a process definition;
- Orchestrating the execution of all Local Business Processes;
- Gathering and storing all (global and local) monitoring details during execution;
- Informing the user of the status of execution (at global and local level);
- Analysing and passing any control orders from the user to the Local Business Processes.

The Global Enactment module uses two kinds of input. The first is a verified and validated global process definition expressed in a BPEL process file. The second is a WSDL interface (Web Service Interface) file which describes how the global process can be accessed by the outside world. The module also produces two kinds of output: SOAP messages for the invocation and orchestration of all Local Business Processes (Web Services) and XML files containing structured information regarding the status of the execution at both global and local level.

The Global Enactment module requires a user interface for two reasons. Firstly, it requires forms for the instantiation (and update, if necessary) of all global

variables required during execution. Secondly, it needs an interface for feedback messages in case of errors or exceptions with proposed actions for the user. For the graphical view of the status of the execution of the global workflow (expanded with local workflows) there is no user interface in the Global Enactment module. Instead, all information regarding the status of execution is transferred to the Global Monitoring UI module which is responsible for depicting the status of execution. The same holds for the graphical interface that enables the user to start and intervene in execution, which is also incorporated in the Global Monitoring UI module.

The Global Enactment module interfaces with several other modules. It interfaces with the Workflow Composition module to allow the transportation of the composed, finalized and verified Global Business Process to the enactment engine that will execute it. The interface with the Local Enactment module allows the invocation and, in general, orchestration of all local business processes by the Global Enactment module. This interface also allows the transportation of all monitoring details between the global and the local level of execution. The interface with the Global Monitoring UI module allows the transportation of all structured monitoring information to the UI that will depict the current status of execution in a graphical way. Moreover, this interface is used for enabling the user to instantiate the execution of a global process definition as well as controlling the execution.

The Global Enactment module relies on the following technology for implementation:

- *Java Platform, Enterprise Edition (J2EE)*: This is a programming platform – part of the Java Platform – for developing and running distributed multi-tier architecture Java applications, based largely on modular software components running on an application server. J2EE includes several API specifications, such as JDBC, RMI, e-mail, JMS, Web Services, XML, and defines how to coordinate them. J2EE also features some specifications unique to J2EE for components. These include Enterprise Java Beans, servlets, portlets, JavaServer Pages and several Web Service technologies. This allows the developer to create an enterprise application that is portable between platforms and scalable, while integrating with legacy technologies.
- *ActiveBPEL*: The ActiveBPEL open source engine is a commercial-grade open source implementation of the Business Process Execution Language for Web Services Version 1.1 specification, and is fully compliant with that spec.
- *Web Services Protocol Stack*: According to the W3C, a Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface that is described in a machine-processable format such as WSDL. Other systems interact with the Web Service in a manner prescribed by its interface using messages, which may be enclosed in a SOAP envelope, or follow a RESTful approach. These messages are typically conveyed using HTTP, and normally comprise XML in conjunction with other Web-related standards.

Figure 8.6 shows the internal architecture of the Global Enactment module. The heart of the module is the "BPEL engine" discussed earlier. Coupled to it is the "Monitor and Control" submodule. This submodule extends the functionality of the "BPEL engine" with monitoring capabilities. This is required, as BPEL files are by definition stateless and most commercial BPEL engines do not keep track of the execution of a business process. In addition, the "Monitor and Control" submodule is responsible for monitoring and keeping track of all major events/activities that take place during the execution of a global process definition, at both a local and global level. The "BP-WS" interface provides the connection to the Local Enactment module, using the BP-WS paradigm discussed earlier.

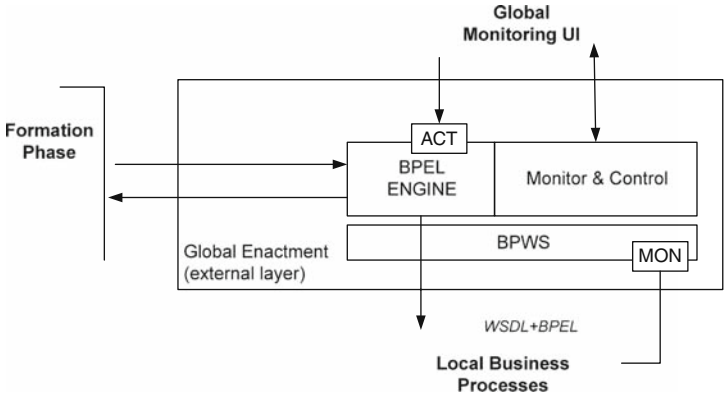

Fig. 8.6 Global enactment module internal architecture

8.3 Local Workflow Enactment

The Local Enactment module is a runtime environment that is capable of executing local business process definitions developed and managed by proprietary WorkFlow Management Systems (WFMS), or other tools, owned by the members of an IVE. Moreover, the Local Enactment module is responsible for informing the Global Enactment module about the status of the execution of the local process definitions. In more detail, this includes the following functions:

- Executing an instance of a local process definition;
- Informing the local supplier of the status of execution (at local level);
- Informing the Global Enactment module of the status of execution (at local level);
- Invoking local Legacy Systems where necessary (as described in the local business process).

The Local Enactment module uses a number of inputs. SOAP messages are used to trigger the execution of a local business process. SOAP messages allow an external party to request the specification/definition of a local business process. SOAP

messages also allow an external party to issue control orders (stop, pause, resume) to a local business process being executed. XML documents are used to represent the legacy system (e.g. EIS) response to the execution of a requested operation by the local business process. The Local Enactment module produces three kinds of output; SOAP messages report the status of execution of a local business process, BPEL files are used to describe the specification/definition of a local business process and XML documents are used to represent a request to a legacy system (e.g. EIS)

Like the Global Enactment module, the Local Enactment module requires two user interfaces: forms for instantiation and update of local variables required while executing a process, and feedback messages in case of errors or exceptions. In general, the user interface of the Local Enactment module is highly dependable on the interfaces provided by the proprietary local tools (e.g. WFMS) which are responsible for executing the local business processes.

The Local Enactment module interfaces with two other modules. The interface with the Global Enactment module allows the invocation of a local business process by the Global Enactment module. This interface also allows the transportation of all monitoring details between the global and the local level of execution. The interface with the legacy systems allows the connectivity and management between the Local Enactment module and heterogeneous EISs, which provides the information infrastructure for an enterprise.

The implementation of the Local Enactment module relies on Java and Web Service technology as discussed for the Global Enactment module. In addition to this, it relies on specific WFMSs, i.e. systems that help organizations to specify, execute, monitor and coordinate the flow of work cases within a distributed office environment. In the CrossWork prototype implementation, the EXODUS i.Perform product has been used for this purpose. With i.Perform, a business can manage and monitor its processes with a view to maximizing profitability (for more information see www.exodus.gr).

As shown in Fig. 8.7, the internal architecture of a Local Enactment module consists of two submodules. A local WFMS is used as a runtime environment that is capable of executing local business process definitions. An example of such a system is EXODUS's i.Perform. The Local WFMS interfaces to the Global Enactment

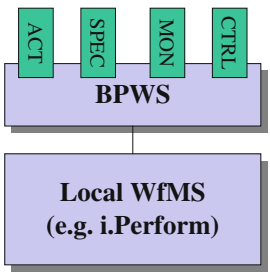

Fig. 8.7 Local enactment module internal architecture

module through a BP-WS interface [7]. As discussed before, the approach of a BP-WS supports an internal business process specification that can be accessed externally through a number of dedicated Web Service interfaces. A BP-WS has the following interfaces (in order of typical use):

- *Invoke and reply interface*: These are not different from the situation where the BP-WS would have been a traditional, black-box Web Service. We cluster them in an activation (ACT) interface.
- *Interface to obtain the business process model*: Through the SPEC interface, a consumer can obtain a process specification of the business process service, e.g. in BPEL. The interface can be considered a reflection interface, as it provides information regarding the behaviour of a BP-WS.
- *Interface to monitor the execution of the internal process*: Through the MON information, a client can obtain status information about the execution of the local process.
- *Interface to control the execution of the internal process*: Through the CTRL interface, a consumer can issue control primitives to influence the execution of a service executed on its behalf; typical control primitives are *pause process*, *resume process* and *abort process*. Invocation of control primitives is typically based on information obtained through the MON interface.

8.4 Coupling to Legacy Systems

In order to complete the enactment process of an IVE, EISs and legacy systems of the IVE members have to be taken into account. EISs provide the information infrastructure critical to the business processes of an enterprise. EIS and legacy applications include relational database systems, enterprise resource planning systems and mainframe transaction processing systems. Legacy systems contain the business logic of an organization, and represent many years of coding, developments, enhancements and modifications. Nevertheless, they are often undocumented, tightly coupled, relatively inflexible and consequently, difficult to integrate into composite business processes. Because most companies have a significant investment in their legacy infrastructure, management is typically not open to replace those legacy systems. Rewriting or significantly modifying large portions of a legacy environment is neither practical nor realistically accomplishable with reasonable resources. The approach taken in the CrossWork architecture is the integration of the existing legacy systems and EIS by means of providing them with a uniform Web Service interface that can be integrated into Local Workflows.

The Legacy Integration module allows the creation of new value from existing enterprise systems by means of integrating them into new business processes, which can be exposed at local level as well as global level participating in IVEs business processes. CrossWork's Legacy Integration module offers to the Local Enactment module a unified interface to perform transactions into heterogeneous information

systems, hiding the details of the operation in each concrete EIS. Solutions addressing the Legacy Integration such as the CrossWork Legacy Integration module have to consider the following aspects:

- *Heterogeneity and complexity of EIS*: Each EIS has a different programming model, increasing the complexity and the effort required to perform the integration.
- *Transactional access to EIS*: This is needed in order to preserve data integrity for critical enterprise information.
- *Secure access to EIS*: It is critical to assure data integrity in all access to EIS information – any loss of information or unauthorized access could be extremely costly to a company.
- *Scalability*: Companies exist in a very changing world – it is important to provide tools that can grow according to the needs of the enterprise.

The Legacy Integration module includes the following functions. It provides a Web Service interface allowing connecting and executing business processes into multiple and heterogeneous EIS and legacy systems. It validates requests to a concrete legacy system, avoiding unnecessary calls to this system. It transforms both requests and responses to the legacy system, to facilitate the integration with the Local Enactment module.

The Legacy Integration module takes as input SOAP messages describing the business process to execute. It specifies the EIS where to execute the operation, the business process to be executed, and an XML document with the request for this business process. As output, the Legacy Integration module produces SOAP messages representing the system's response to the execution of the requested business process.

As the module has interface functionality only, the Legacy Integration module does not have a user interface. The Legacy Integration module interfaces with the Local Enactment module. Local workflow enactment requests the legacy interface to execute operations on EISs.

The Legacy Integration module relies on the following technology for implementation:

- *JCA*: Employs Java-based technology solution for connecting application servers and EISs as part of enterprise application integration (EAI) solutions. Legacy Integration uses JCA resource adaptors to connect to several EIS.
- *Apache Axis*: Employs Java and XML-based Web Service framework consisting of an implementation of the SOAP server, and various utilities and API's for generating and deploying Web Service applications. Legacy Integration uses Axis in two ways; to provide a Web Service interface to the module, and to develop client classes to use Web Service connectors, mainly used to connect to .Net platforms.
- *EJB*: Employs server-side components that encapsulate the business logic of an application.

- *XSLT*: Employs XML-based language that is used for the transformation of XML documents. Within the Legacy Integration module it is used to transform the input and output messages.
- *XML Schema*: Employs the XML schema that is a description of a type of XML document. In the Legacy Integration module it is used to validate input messages.

As shown in Fig. 8.8, the internal architecture of the Legacy Integration module consists of the following submodules:

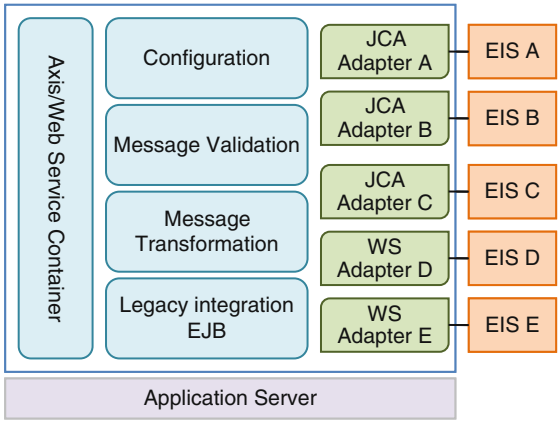

Fig. 8.8 Legacy integration module internal architecture

- *Legacy Integrator EJB*: It contains the business logic of the Legacy Integration module;
- *Configuration*: It stores EIS information, such as operations, users, passwords and servers;
- *Message Validation*: Input messages are validated to XML schemas defined for each EIS and each operation;
- *Message Transformation*: Messages can be transformed using an XSLT defined for each EIS and each operation.

8.5 Related Work

Besides the enactment approach followed in CrossWork, many different approaches exist today with regard to business process execution technologies, architectures, environments and languages. In this section we discuss some of the most important research and industrial efforts in this area.

In CrossWork, we use a Web Service orchestration standard (BPEL) for business connectivity. Although orchestration and its standard seem to be the most powerful tools for enterprise collaboration, there is another way of collaboration designed

to reduce the inherent complexity of connecting Web Services together, called choreography. While with orchestration, the process is always controlled from the perspective of one of the business parties, choreography is more collaborative in nature. Each party involved in the process describes the part they play in the interaction. Choreography tracks the sequence of messages that may involve multiple parties and multiple sources. It is associated with the public message exchanges that occur between multiple Web Services. The underlying intuition behind the notion of choreography can be summarized as "dancers dance following a global scenario without a single point of control".

The respective standard which is designed to reduce the inherent complexity of choreographed Web Services is called Web Services Choreography Description Language (WS-CDL). It is a specification by the W3C [10] defining an XML-based business process modelling and execution language that describes collaboration protocols of cooperating Web Service participants, in which services act as peers, and interactions may be long-lived and stateful. While BPEL allows existing services to be orchestrated into composite services, the WS-CDL goes a step further and describes the relationships between services in a peer-to-peer scenario.

Another relevant area that currently receives attention is the area of semantically annotated business processes. There, the Business Process Management Initiative (BPMI) [3] has developed a standard Business Process Modelling Notation (BPMN) [4]. The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts who create the initial drafts of the processes to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation. BPMN can be used in conjunction with BPEL, or can provide a process notation in its own right. The basic BPMN notation is very similar to that of UML Activity Diagrams. BPMN was developed with the support of many leading BP process modelling tool vendors and is well on the way to becoming the most popular notation among process modellers, although companies use so many different process notations that standardization will likely take quite a while.

With regard to the European funded projects, one of the most promising efforts is the ECOLEAD [5] project. ECOLEAD, among other issues, aims to propose new and strong mechanisms to support collaborative and distributed business process management. In that area, the challenge for ECOLEAD is to find and build approaches that support the management of virtual organizations (quite synonymous with virtual enterprises) from different aspects including enactment. For ECOLEAD, the Supply Chain Operations Reference (SCOR) [9] modelling approach including performance metrics is promising for further use. SCOR is a process reference model that has been developed and endorsed by the Supply Chain Council as the cross-industry-standard diagnostic tool for supply chain management, and describes the business activities associated with all phases of satisfying a customer's demand. SCOR uses graphical decomposition of processes from the Level 1 functions (PLAN-SOURCE-MAKE-DELIVER-RETURN) to Level 3

process elements. Modelling of inter-enterprise processes is undertaken by connecting one process element (i.e. SOURCE) from one supply chain member to the process element (i.e. DELIVER) of another member. Major benefit and reason for widespread acceptance of SCOR are the defined Key Performance Indicators (KPIs) used to measure and compare (benchmark) the performance of companies in their domains.

8.6 Coordinating Work Across the Instant Virtual Enterprise

In this chapter, we have described the CrossWork business process (workflow) enactment subsystem. This system consists of three main modules. The Global Enactment module is responsible for the execution of a global business process within an IVE, i.e. the workflow that links all the local workflows of members in that IVE. The Local Enactment module is responsible for the execution of local workflows within the boundaries of a single IVE member. The Legacy Integration module provides the interfaces to couple local workflow management to legacy systems residing within the boundaries of IVE members. The division of the subsystem into these main modules provides both the separation of concerns needed for a modular implementation and the flexibility to deal with extensions towards the future.

As we have shown in this chapter, the realization of the CrossWork enactment subsystem is heavily based on standardized and available technology. We use the BPEL process definition language, and a stable open source BPEL engine for global process enactment. We have used a specific local enactment engine for our prototype implementation, but it is shielded by well-specified interfaces and can thus be replaced by other choices. For Legacy Integration, we use a number of standard technologies.

Once the processes linking different partners are assembled, we need to deploy them on the existing service-based infrastructures of different members of the VE. A global workflow module is then responsible for coordinating the work access to different workflow engineers, driving the processes across the VE “in sync”.

References

1. ActiveBPEL, Website: www.activebpel.org, 2008.
2. OASIS Web Services Business Process Execution Language (WSBPEL), Version 2.0, May 2007.
3. Business Process Management Initiative, www.bpmi.org, 2008.
4. Business Process Modeling Notation, Version 1.1, http://www.bpmn.org, 2008.

5. European Collaborative Networked Organisations Leadership Initiative, ECOLEAD Project, http://ecolead.vtt.fi/, 2008.
6. Grefen, P., CrossWork Global Architecture, CrossWork Project Deliverable D4.1, 2006.
7. Grefen, P., Ludwig, H., Dan, A., Angelov, S., An Analysis of Web Services Support for Dynamic Business Process Outsourcing, Information and Software Technology, Vol. 48(11), pp. 1115–1134, 2006.
8. Michelson, B., Business Process Execution Language (BPEL), Primer: Understanding an Important Component of SOA and Integration Strategies, 2006.
9. Supply Chain Operations Reference-model (SCOR), Supply Chain Council, http://www.supply-chain.org, 2008.
10. Web Services Choreography Description Language, Version 1.0, http://www.w3.org/TR/ws-cdl-10/, W3C, 2005.