Distributed Processing Environment: A Platform for Distributed Telecommunications Applications

Vendors developing applications for a heterogeneous, distributed environment need to be able to build towards a platform that integrates all the management and control functions of distributed computing into a unified software architecture that allows their applications to be available from any point in the network regardless of the system or geographic location.

by Frank Leong, Satya P. Mylavarabhata, Trong Nguyen, and Frank Quemada

The HP Distributed Processing Environment (DPE) provides infrastructure services that facilitate the rapid development, deployment, and management of distributed applications in the telecommunications arena. DPE is a key component of the Telecommunications Information Networking Architecture (TINA), an architecture for multimedia networks that emphasizes distribution and interoperability of telecommunications applications. TINA is an evolving architecture and is governed by the TINA Consortium (TINA-C), which is a project sponsored by 40 leading telecommunications and computing companies. The project's aim is to find a way to integrate all telecommunications management and control functions into a unified logical software architecture supported by a single distributed computing platform.

This paper describes the architecture and components that make up HP DPE, a product that is compatible with (and will evolve with) the TINA specifications.

INA, TINA, and DPE

HP DPE and TINA have a common root in the Information Networking Architecture (INA), which was originally developed at Bellcore. TINA's architecture specifies a distributed processing environment based on the original INA DPE specifications. HP DPE provides key infrastructure services for INA and TINA.

INA defines a methodology and framework for developing, providing, and maintaining highly distributed systems, characteristic of the next generation of communications environments. INA leverages and combines the efforts of multiple standards bodies, research organizations, development organizations, and consortia (e.g., TMN, OSCA, OSF/DCE, OMG CORBA, OSI/NMF, etc.). Fig. 1 shows the relationship between INA DPE and the TMN (Telecommunications Management Network) model. TMN is described in ***Article 1*** and the DPE services are described later in this article.

INA applications and services are deployed as software modules called *building blocks*. A building block is made up of several objects and can be installed and modified independently of other building blocks in the network. Building blocks interact with one another via interfaces called *contracts*. Contracts are the exposed interfaces of an object in that they are used for communication between building blocks. They are also backward compatible to ensure interoperability between software objects contained in multivendor building blocks. Contracts are subject to authentication and access control checks.

A building block can be a server or a client or both. A server must offer one or more contracts to allow clients to interface and make use of its services. In the DPE architecture (described below), applications are modeled as building blocks. The DPE itself is made up of server building blocks (e.g., contract trader, repository, etc.) which offer contract interfaces to application client building blocks.

The INA structure enables distributed software building blocks from multiple suppliers to interoperate. This distributed object computing results in faster software development since there is greater software reuse and modularity in design.

In summary, INA is a framework for interoperability, portability, and network resource management. The following goals have been established for INA:

- Rapid and flexible introduction of new services
- Reuse of software modules
- Use of general-purpose solutions
- Multivendor hardware and software solutions
- Independence of applications from the transport implementation technology

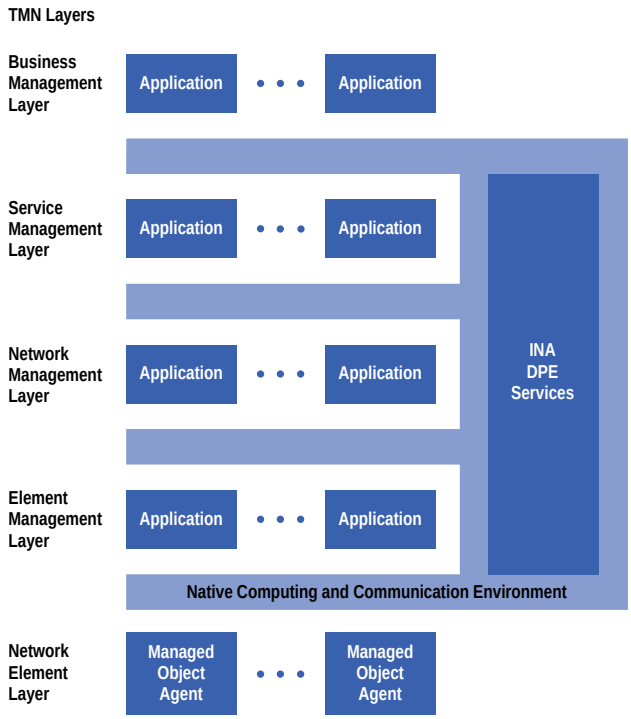

Fig. 1. *The INA DPE architecture applied to the Telecommunications Management Network (TMN).*

- Separate transport technologies from higher-level control and OAM&P (operation, administration, maintenance, and provisioning)
- Allowance of customer access to OAM&P services
- Seamless integration of services
- Network and element management.

DPE Architecture

Fig. 2 shows the components and services that make up the DPE architecture.

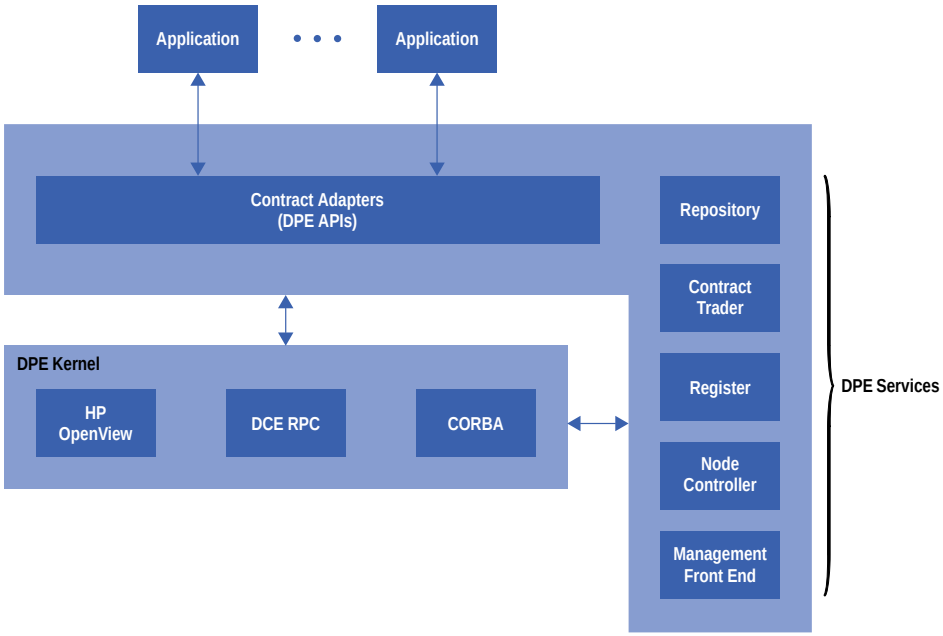

Fig. 2. *Components of the DPE architecture.*

DPE Kernel. The DPE kernel provides the foundation for building block interaction and execution services. To implement these services, the DPE kernel uses the services provided by the underlying native computing and communications environment, which include:

- DCE: threads, security, RPC, and IDL compiler
- CORBA: HP ORB+ with IIO and DCE CIO protocols and C-IDL compiler
- HP OpenView components: XMP API, pmd (postmaster daemon), orsd (object registration service), and ovead daemon (event sieve agent).

The DPE kernel is resident in every node of a distributed system. Building blocks and other DPE components at a node cannot access the DPE kernel at other nodes directly. Access to the DPE kernel services at a remote node is accomplished using the interprocess communication facilities of the native computing environment of the node.

Contract Adapter. A contract adapter is an application programming interface that provides all the transparencies required by a client or server building block. It also provides an API for accessing either application-level services or services provided by DPE. Contract adapters are kept as library modules which can be linked with building blocks before or during execution.

The inclusion of adapters as components of DPE implies that the components of DPE increase over time as new applications are deployed in a network. When a contract type is specified and registered for some application-level service, adapters for these contract types can be automatically generated and made a part of DPE.

DPE Services. Each DPE service is a building block and access to its functions is only through contracts offered by the DPE service. A node may have zero or more DPE services installed. Since access to a function provided by a DPE service is available only through a contract, a building block or a DPE service in a node can use the functions provided by a DPE service in a remote node. Thus, DPE services depend on the communication and execution services provided by the DPE kernel. References to contracts of some of the DPE services, such as the trader, can be passed to a building block when it is activated.

Although both DPE services and applications are built using the concepts of building blocks and contracts, there is a fundamental difference between the two. DPE services do not provide network resource management functions, nor do they provide telecommunications services to network customers. These functions are provided only by applications.

Fig. 3 shows the interactions among the DPE services shown in Fig. 2. An arrow directed from one service to another indicates that the source service provides services to the destination service.

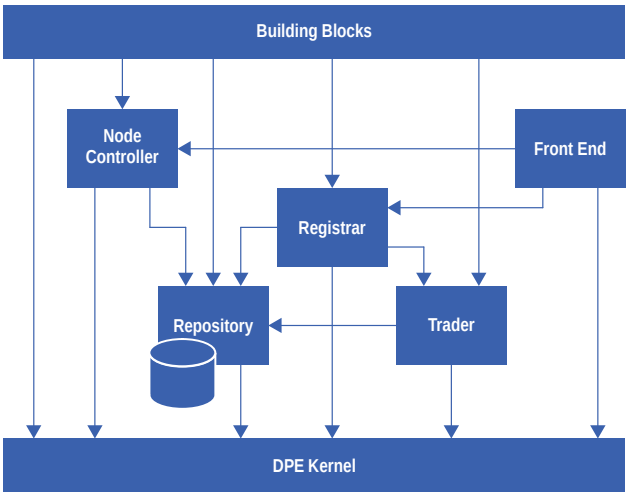

Fig. 3. *Interrelationships between different components in the DPE services.*

Contract Trader. This DPE service provides a discovery service for client and server building blocks. It is the key service for providing location transparency in a distributed network. When a building block offers a contract, information about this contract is conveyed to the DPE kernel. This information includes the name of the corresponding contract type and the value of the service attributes provided by the contract. DPE stores this information in the repository.

When a client wishes to invoke an operation defined in a specified contract type, it queries the DPE trader for one or more references to contracts that match the specified type and whose service attribute values satisfy a constraint expression supplied by the client. Regardless of where the server is physically located, the client can discover servers at run time, based on the latest contract information recorded in the repository database. The DPE trader provides two types of contract trading: attribute-based trading and resource-based trading.

The attribute-based form of contract discovery is based on the specified contract type and a constraint expression involving any number of the service attributes. The constraint expression used by HP DPE is modeled after the ANSAware 2.0

constraint language. This language supports relational operators on attributes and maximum, minimum, and logical operators. This provides a great deal of flexibility in how a client discovers a server.

An example of a constraint expression might be a request to find one or more print servers that can print in color, provide A4 size paper, and use PostScript™ fonts. The constraint language would express this request as: attribute_list = color, A4, postscript. If we need a certain capacity and speed for the printer, we might add a request for faster than six pages per minute: attribute_list > 6.

A resource-based form of contract discovery is an extension of attribute-based trading and is used by resource management applications. In resource management applications, it is typical to provide service over a domain of resources. This domain may be dynamic. An example would be a connection management application that is responsible for providing connection management services to all clients whose phone numbers (domain) begin with area code 408 and have the exchange number 447. This application may offer contracts over a domain that may vary in size depending on how many phone numbers are actually assigned (e.g., all the numbers following 447). This type of trading requires the client to supply a contract type name, a constraint expression, and the name of the resource. With this information the HP DPE trader can locate a server offering a contract of the appropriate type that satisfies not only the search constraint expression, but also the specified resources.

Repository Server. This DPE service maintains persistent information for the operation of DPE. It stores specifications of trading attributes, contracts, building blocks, and configuration information. The repository server provides operations for the creation, retrieval, update, and withdrawal of DPE-persistent objects. These reference objects are used to initialize, activate, deactivate, and withdraw contract and building block instances using a generic front-end administrative tool. This server is implemented using the ObjectStore 4.0 OODBMS from Object Design Inc.

The information stored in the repository can be used for several purposes. The DPE front end can traverse repository information to help application developers locate potential reusable attribute types, contract types, and building-block type specifications. It also provides type information that allows the DPE controller to check for valid operation parameter types at run time. The following three kinds of information are stored in the repository:

- Specification information. This consists of information contained in contract type specification templates and building-block type specification templates registered with the DPE repository.
- Configuration information. This consists of information contained in the building-block configuration templates, contract configuration templates, and node configuration templates registered with the DPE repository. This means that the repository contains information needed for managing building-block instantiating operations or startup operations.
- Trading information. This consists of information that supports trading operations, specifically contract types and contract instances.

Registrar. This DPE service provides registration and withdrawal services for the various templates used in the operability services, including specification templates, installation templates, and configuration templates. Its function is to parse and verify the correctness of the specification templates before invoking the registration operation of the repository server.

Node Controller. The node controller at each node provides activation, deactivation, monitor, and restart functions for building blocks configured in that node. It receives notifications when a building block is started and deactivated, and continuously monitors the "liveness" of all building blocks executing in the node. Since the implementation of these functions is dependent on the native computing environment's facilities, one instance of the node controller building block is required in each node.

Management Front End. HP DPE provides a graphical front end and a command line interface to DPE system administration, building-block management, repository browser functionality, and DPE shutdown and restart functions. This user interface offers a generic and uniform way of managing the whole DPE domain from any node. DPE objects present in the GUI are organized in a hierarchical structure similar to the renowned Smalltalk browser. This structure is organized as nodes, building block types and instances, and contract types and instances (see Fig. 4). The DPE front-end interface provides the following functions:

- Contract building-block type registration
- Activation, shutdown, and withdrawal of building-block instances
- Activation, shutdown, and withdrawal of contract instances
- Setup and modification of contract trading attributes
- Browser for DPE objects.

With the command line interface, routine DPE administrative tasks can be automated using shell script languages.

DPE Telecommunications Examples

This section provides two examples of the use of HP DPE in the design and deployment of telecommunications services and applications. The steps illustrated in these examples present a high-level view of the communications that occur. The actual designs are much more complex. Also, to reduce the complexity of the figures, three assumptions have been made:

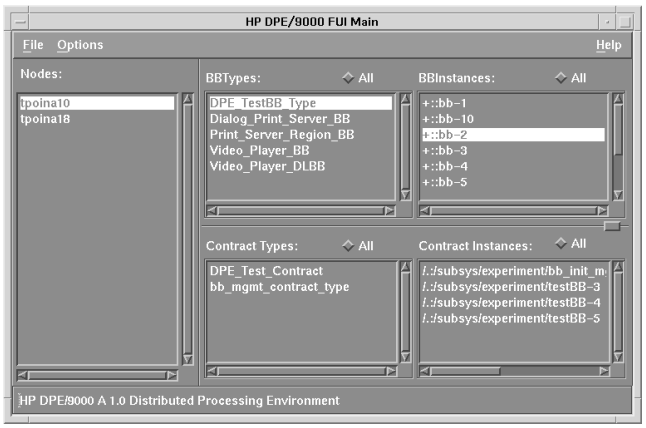

Fig. 4. *The DPE graphical user interface.*

- All interfaces that are used have already been registered with the DPE registrar, and binding information for each interface is available from the DPE repository.
- All communication with the DPE trading service is done via an RPC mechanism.
- Most applications will either trade at initialization time to obtain binding handles or simply use a well-known address to maximize throughput. Trading during execution will most likely be reserved for those occasions that dictate the need for dynamic binding. For illustrative purposes, however, the examples show trading occurring for each initial communication between any two modules.

Example 1: Permanent Virtual Circuit Service

The most basic connection service provided by broadband networks is a permanent virtual circuit (PVC) service. This service provides the capability of setting up a connection between two or more points with given bandwidth and quality-of-service (QoS) parameters. Typically PVCs are long-term connections used to interconnect LANs or provide long-term video service between distant points. Fig. 5 illustrates how a simple PVC service might be designed using an architecture based on INA. Each of the following steps corresponds to a number in Fig. 5.

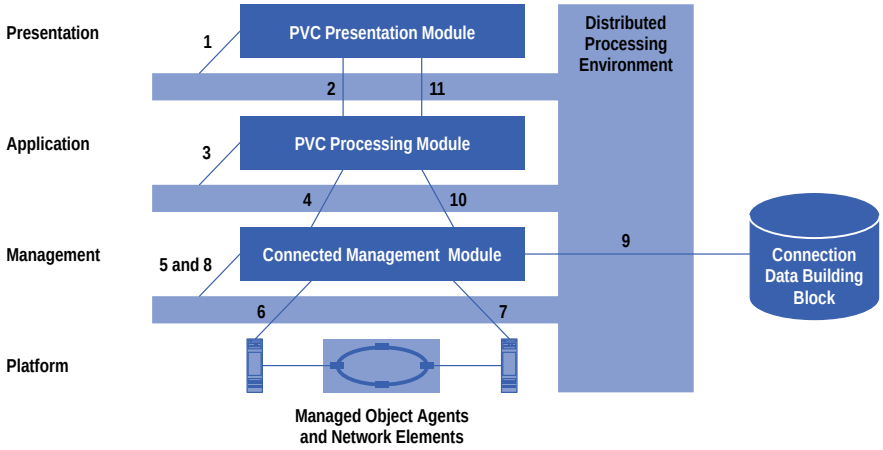

Fig. 5. *The architecture for a permanent virtual circuit service.*

1. The PVC presentation module consults with the DPE trading service for the location of the PVC processing module applications. This communication is done via an RPC (remote procedure call) interface.
2. The PVC presentation module provides the PVC processing module with the user input parameters that define the PVC being requested. This communication is done via an RPC interface.
3. The PVC processing module consults with the DPE trader to locate the connection management application server that controls the switch servicing the originating end of the PVC. This is done via an RPC interface.
4. The PVC processing module uses the DPE RPC mechanism to access the connection management application. If the connection requires more than one switch, the connection manager will trade for and bind to another connection manager to move the connection towards the termination point (this is not shown in Fig. 5).

5. The connection manager trades for the binding handle of the managed object agent that services the originating (and terminating if local) points. For performance reasons, in most designs this step is done at system initialization time.
6. The connection manager instructs the managed object agent to connect the originating end using the DPE system management protocol CMISE (Common Management Information Service Element).
7. The connection manager instructs the managed object agent to connect the terminating point using the CMISE protocol.
8. The connection manager uses RPC to request a binding handle from the connection data building block.
9. The connection manager requests the connection data building block to update its data store to reflect the addition of the new PVC connection. The communication is done via RPC.
10. The connection manager reports the establishment of a connection back to the PVC processing module via an RPC.
11. The PVC processing module returns the status of the connection establishment back to the PVC presentation module for display to the user.

Example 2: Switched Virtual Circuit Service

This example shows that the modularity and code reuse capability of the DPE architecture can be used to add new features. The switched virtual circuit implementation shown in Fig. 6 provides users with the capability to establish or reconfigure existing connection sessions at any time, much like voice telephony service. As shown in Fig. 6 the connection management, data building block, and managed object agents are all being reused. Only the top two modules need to be replaced with new code.

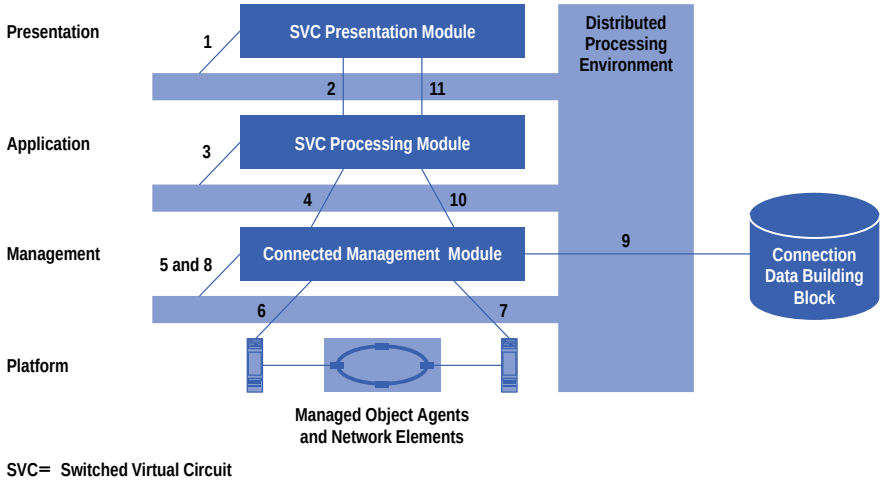

Fig. 6. *The architecture for a switched virtual circuit service.*

Summary

This paper has presented an overview of the HP DPE implementation. DPE plays a key role within the Telecommunications Information Networking Architecture (TINA). HP DPE offers a development environment to develop distribution transparency for both RPC-based and CMIP-based INA-compliant applications. This paper has also detailed the services provided by HP DPE and described the implementation of the contract trading servers and contract adapters, the key components providing distribution transparency.

Acknowledgments

The authors would like to acknowledge other members of the development and product team: Joel Fleck, Bruce Greenwood, Hai-Wen Liang, David Wathen, and Chris Liou.

PostScript is a trademark of Adobe Systems Incorporated which may be registered in certain jurisdictions.

▶ Go to Article 3
▶ Go to Table of Contents
▶ Go to HP Journal Home Page