Chapter 2

The World-Wide Virtual Machine

The World-Wide Virtual Machine (WWVM) is a software system that connects a heterogeneous collection of networked computational resources and provides users with a view of a single metacomputing platform. It naturally supports both message-passing and dataflow computing paradigms. Web technologies combined with HPCC technologies form the basis of the WWVM architecture. This chapter describes that architecture and the implementation details.

2.1 Motivations

Recent advances in both network bandwidths and microprocessor performance are radically altering high-performance computing environments. There is a strong trend toward building virtual computing platforms [GNW, RYH 94] from heterogeneous resources distributed on a network.

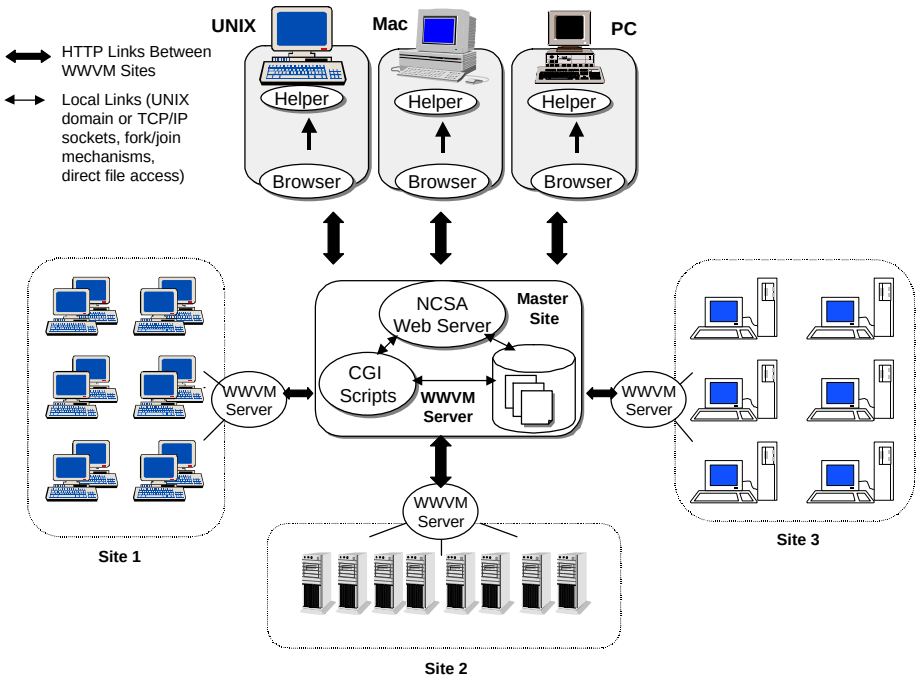

Figure 2 1. Overview of the client-server relationship in the WWVM architecture.

Such virtual computers with their enormous computing power and storage capacities can be used to attack many computationally challenging problems such as the Grand and National Challenges [HPCC 96, Gran 91], whose computing and storage requirements are beyond the capacity of a single dedicated parallel (or super) computer.

Grand Challenges are major problems of science and society whose solutions require a 1000-fold or greater increase in the power and speed of supercomputers and their supporting cast of networks, storage systems, supporting software, and virtual environments. The emerging World-Wide Web (WWW) infrastructure is expected to make great contributions to the building of virtual computing environments that will allow large computational problems to be solved more cost effectively by using the aggregate power, features, and memory of many network-

connected computers. The WWW brings to every type of machine in the world a standard open interface through which we can manipulate individual machines on a network.

The WWVM exploits remote resources on the Internet or on local or wide-area networks in order to provide a seamless virtual metacomputing environment (Figure 2-1). It may contain different types of architectures and is very suitable for solving parallel problems as well as *metaproblems* consisting of many constituent subproblems, each of which is suitable for a specific type of computer architecture. Yet, the metaproblem as a whole is outside the scope of a single computer architecture. It is important to run the load on the most suitable computer for a particular problem in order to use such an environment as efficiently as possible.

Modified Parallel Virtual Machine (PVM) [GBD+ 93] daemons, along with Common Gateway Interface (CGI) [CGI 96] extended Web servers [HTTPD 96], form the basis of the WWVM's communication layer. The communication layer supplies the functions to automatically start up tasks and coordinates communication and other functions between tasks. It also takes care of necessary data conversions from computer to computer, as well as low-level communication issues. The user perceives a single, unified system, and the details of the network and individual machines that make up the virtual machine are not directly visible. CGI modules construct the coordination and configuration engines, while a Web-based interface (accessible by using Web browsers) constitutes the front end of the machine. Web servers not only may become computation nodes of this machine, but may also behave as an interface to other machines located behind firewalls in the same organization performing the configuration operations on their behalf, so that they can be included in the WWVM configuration.

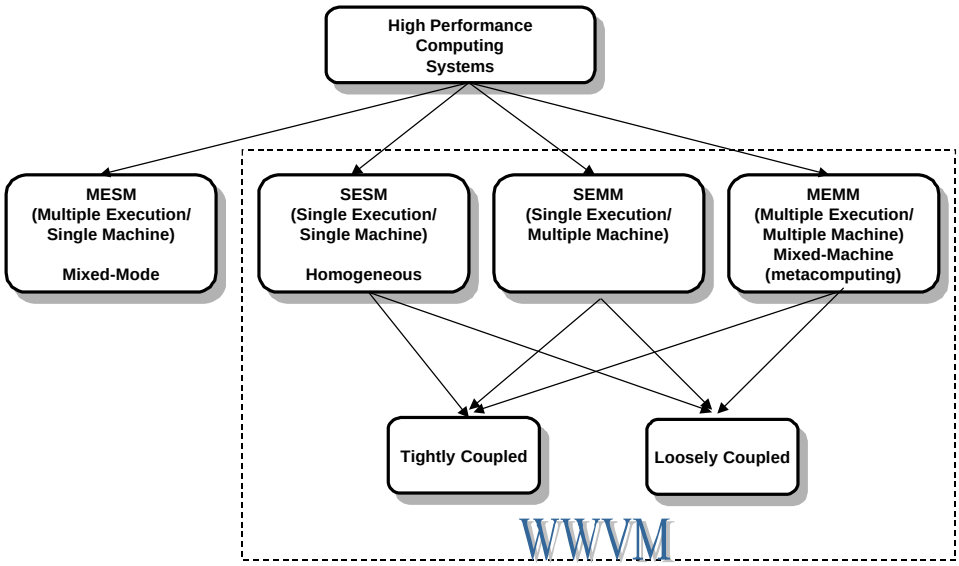

Figure 2 2. A taxonomy of the computing systems according to execution mode and machine model.

2.2 A Taxonomy of Computing Systems

Ekmecic et al. [ETM 95] presents a taxonomy to distinguish computing systems according to the supported execution modes and the number of presented machine models. The type of parallelism supported by a machine represents the *execution mode (EM)*, whereas the machine architecture and processor speed define the *machine model (MM)*. For example, scalar, vector and dataflow processing represent different execution modes. A Sun SPARC and an SGI Challenge, or two Sun workstations with different processor speeds, are considered to represent different architectures.

According to these two criteria, computer systems can be divided into four disjoint classes, as shown in Figure 2-2:

1. SESM (single-execution mode, single-machine model) – homogeneous systems without any elements of heterogeneity, such as uniprocessors and many of the parallel or distributed systems (massively parallel processors (MPPs), homogeneous workstation clusters.)
2. SEMM (single-execution mode, multiple-machine models) – quasi-heterogeneous systems in which different nodes represent different architectures and speeds, but still support only one execution mode. Examples include [AVA 92, Haddad 94, SchH 94].
3. MESM (multiple-execution modes, single-machine model) – same as the classical mixed-mode heterogeneous systems.
4. MEMM (multiple-execution modes, multiple-machine models) – same as the classical mixed-machine heterogeneous systems.

It should be noted that Freud and Siegel [FreS 93] earlier defined the MESM and MEMM as mixed-mode and mixed-machine heterogeneous computing systems, respectively. *Heterogeneous computing systems* provide a variety of architectural capabilities, orchestrated to perform an application whose subtasks have usually diverse execution requirements.

A *mixed-mode* heterogeneous computing system is a single parallel-processing machine that is capable of operating in either the synchronous SIMD or asynchronous MIMD mode of parallelism, and can dynamically switch between modes at instruction-level granularity with generally negligible overhead [FCS 91]. Most famous examples of mixed-mode systems include PASM [Siegel 86], TRAC [Sejn 80], OPSILA [AugB 86], and Triton/1 [Herter 93].

A *mixed-machine* heterogeneous computing system is a heterogeneous suite of independent machines of different types interconnected by (usually) a high-speed network. Unlike mixed-mode machines, switching execution among machines in a mixed-machine system requires measurable overhead, because data may need to be transferred among machines. In mixed-machine systems, the set of subtasks may be executed as an ordered sequence and/or

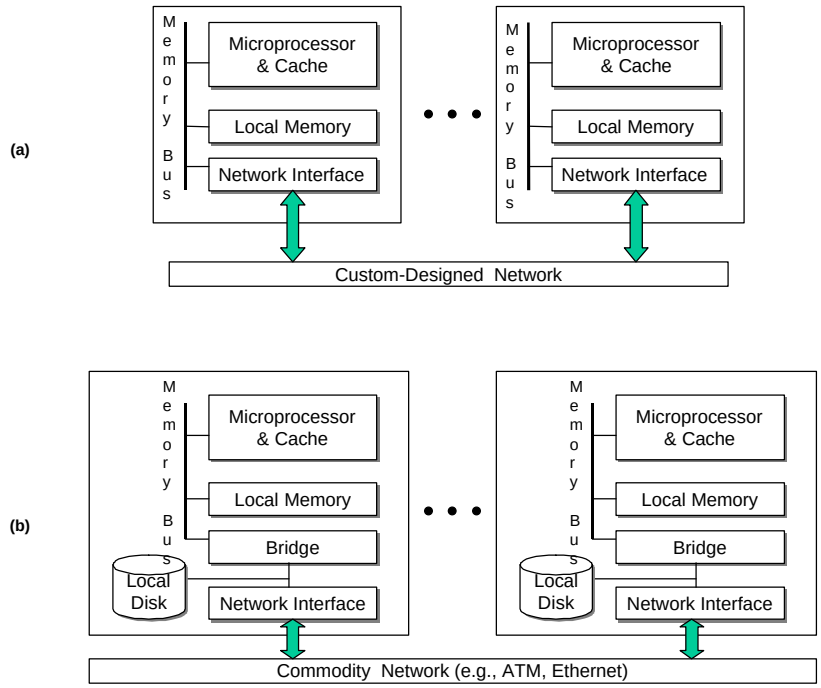

Figure 2 3. A comparison of the (a) MPP and (b) cluster of workstations architectures.

concurrently on multiple machines. Examples of mixed-machine systems include the ones given in [KMC 93, MFS 94, ScoP 94], Nectar [Arno 89], and any architecture that might be lying under software systems such as PVM [Sund 90], SmartNet [Freu 93], Mentat [Grim 91, Grim 92, Stra 92], and Schooner [Homer 91, HomS 92, HomS 94].

Heterogeneous computing systems are also distinguished by the manifestation of heterogeneity: temporal and spatial [FreS 93]. *Temporal manifestation* of heterogeneity implies that the heterogeneous system is executing in one mode at one moment and in some other mode at some other time. *Spatial manifestation* of heterogeneity means that a heterogeneous system supports different execution modes at the same moment, but at different machines. The

heterogeneity of execution modes can be manifested in both the temporal and spatial domains. The heterogeneity of machine models is manifested exclusively in the spatial domain.

Mixed-machine heterogeneous computing has also been referred to as *metacomputing* [KPS+ 93, KMC 93]. The concept of metacomputing is relatively recent [NSF 92, Burns 92, Jova 92, Mars 92, NCSA 91] and is reflective of the rapid advances in computer hardware and networks. In 1992 the National Science Foundation (NSF) accepted a proposal to integrate the four NSF-sponsored supercomputer centers into a single metacomputing facility. The central concept of the metacomputing proposal was that problems should be able to migrate to the appropriate computer architecture(s) without regard for where the computers are physically located.

Incorporating many different types of computers to achieve a task is a time-consuming and tedious job if done manually. The specific benefits of various architectures (scalar, vector, SIMD, MIMD, etc) should be leveraged to address complex grand challenge problems by providing the capability to implement distributed solutions in a transparent manner. Scientists require access to several computers to solve complex problems, yet they want to spend their time doing science. They are generally not interested with details of computers and accompanying technologies, therefore the metacomputer should be simple to use.

The WWVM can be considered as an MEMM or a *metacomputer* for solving metaproblems. The nodes of the WWVM can be a single supercomputer, a workstation farm, or even another WWVM configured as a message-passing machine. When WWVM includes only a set of identical workstations in its configuration, it behaves like a SEMM system; when the workstations are from different vendors, it is representative of an SESM system.

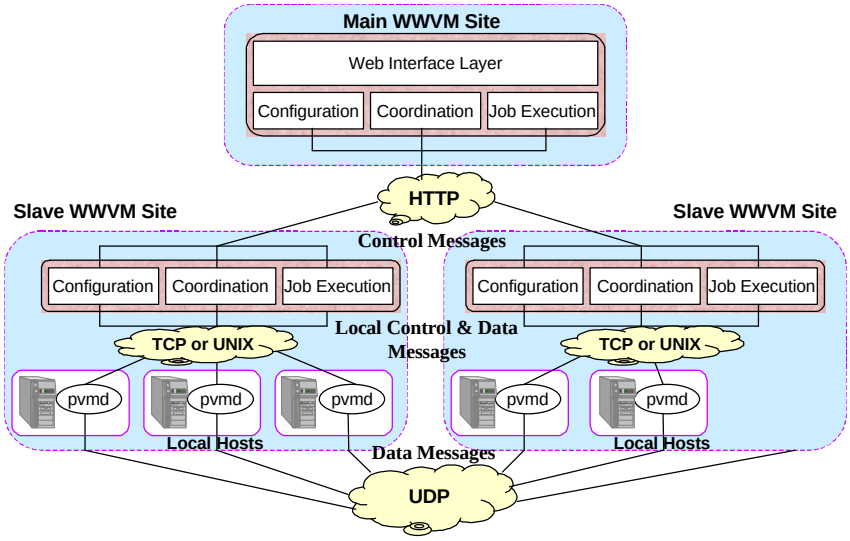

Figure 2 4. System architecture of the virtual machine.

The last two configuration modes (i.e., SEMM and SESM) are usually known as *cluster computing systems*. Cluster computing refers to a cluster of workstations connected through a low-cost commodity network such as Ethernet, FDDI, or ATM, and has become an attractive alternative parallel-computing platform. The total computational power of such an assembly of machines can be enormous. In contrast to a tightly- coupled network interface that is connected to the memory bus of a node in distributed-memory parallel computers, the network interface is loosely coupled to the I/O bus in a workstation cluster. Each node of a cluster of workstations is a complete workstation minus the peripherals except for a local disk. As shown in Figure 2-3, a complete workstation Unix operating system resides on each node, as compared to some MPPs where only a microkernel exists. Currently, clusters of workstations are more useful for attacking

problems of larger granularity than the MPPs with custom-designed fast interconnections; however, as faster network technologies develop, this difference tends to fade away.

2.3 System Architecture

WWVM design has a modular structure with three layers: the Web interface layer, the middle layer, and the virtual machine layer (Figure 2-4).

The Web interface layer provides a link between the application programmer and the WWVM server site. The middle layer serves as "glue" between the Web interface layer and the virtual machine layer. It is the core of the WWVM operation. The virtual machine layer combines the WWVM servers on the Internet or other machines coordinated by those servers (if any) to form a parallel virtual computer. Below, the functions and implementation of each layer are explained in detail.

2.3.1 Web Interface Layer

The *Web interface layer* consists of a Web-based graphical user interface (GUI) and interface layer modules. The user may choose computational services, supply program code, and configure the WWVM using Netscape or another JavaScript- and Java-enabled Web browser of choice. The GUI is implemented using Web/HTML forms with JavaScript functions and Java applets. The interface layer modules handle the communication between the WWVM server site and the GUI displayed through the Web browser. They translate users' requests and responses into internal data structures, and replies of the server into a form that can be displayed to the user in a meaningful way. In short, this layer provides a link between the application programmer and the WWVM server site. A screen snapshot of the WWVM browser display for configuring of the WWVM by adding machines from the selected two hosts is illustrated in Figure 2-5.

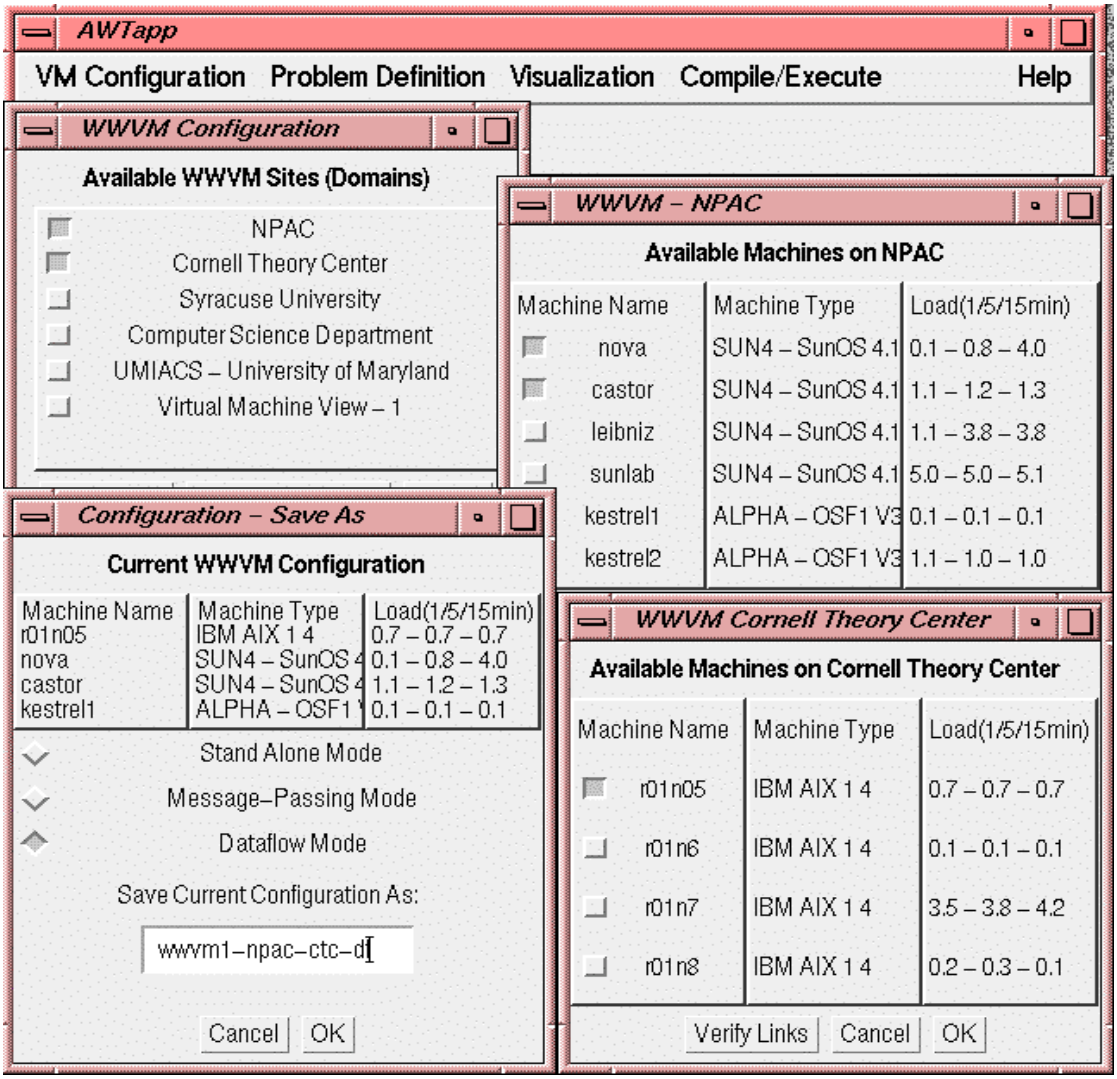

Figure 2 5. A snapshot of the Web front-end for WWVM configuration services.

Users can list the files in their WWVM accounts on different sites or edit files through an editor embedded in the Web-based front end. They manipulate pushbuttons and check boxes to designate the type of services to be performed, such as compiling and linking node programs with runtime support libraries, and then executing the resultant code. They can also specify the service

parameters to be used, such as the platform to be used for compiling and executing the programs or the number of nodes to be used during execution.

The Web interface layer also supports options such as adding or deleting new nodes and forming a user-specific virtual machine configuration. Users can browse through the Web sites volunteering to become part of the virtual machine and add some of them to the current configuration. In fact, these Web sites are the ones with which they are known to be collaborating and for which they have a special Web password.

When a metaproblem is to be defined, the user is presented with a "metaproblem description editor" with which to specify the tasks of a metaproblem and the dependencies (i.e., data and control relationships) among them. Individual tasks may be written in any language (and in any programming paradigm) supported by the platform that task will later be assigned to.

Since Chapter 3 concentrates on the Web-based graphical user interface design and implementation, this subject is not further elaborated on here.

2.3.2 Middle Layer

The *middle layer* consists of three subsystems: configuration, coordination, and job execution. The *configuration subsystem* is responsible for communicating with the virtual machine layer components and initiating and changing the configuration of the virtual machine. It gets the resource requirements (such as the required number of nodes to execute the given program) from the interface layer routines. It also determines current workloads of connected nodes and helps users choose less heavily loaded nodes automatically or semi-automatically. It may initiate new computation nodes to allow scheduling of new tasks to them by the virtual machine layer.

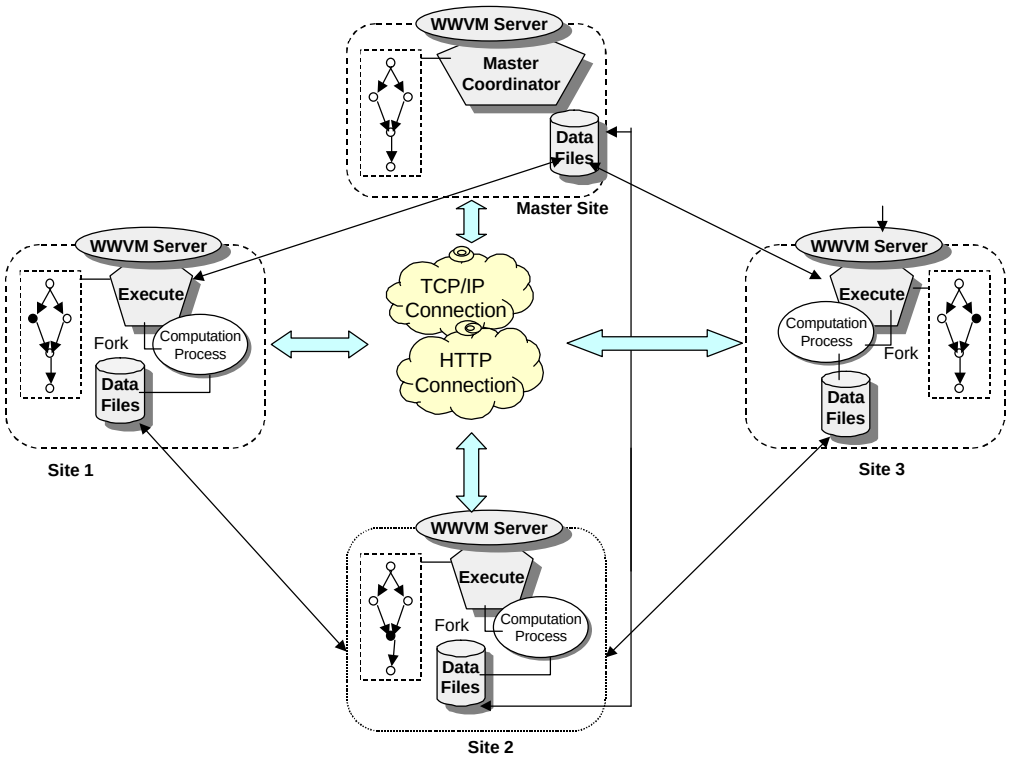

Figure 2 6. Dataflow Architecture of the WWVM.

There are two different ways to select the computational nodes on which tasks will be launched. The user either specifies each and every machine that will become a computational node of the virtual machine, or indicates specific architecture types on which particular tasks will execute.

The middle layer's main role is the processing of compilation and linking requests through its *job execution subsystem*. A series of CGI scripts are executed to carry out the requested compile/link/load/execute cycle, and executable files are placed in machine-specific directories accessible to participating computation nodes. Customized *makefile*s supplied by the user can be

used for the compilation. The underlying virtual machine layer notifies the compilation subsystem about the status of dispatched tasks upon the normal or abnormal termination, and the middle layer presents the result of compilation back to the user through the interface layer. The user can then alter the input program (to correct bugs, perhaps), select a new code, or change the service options before resubmitting the form for recompilation.

The *coordination subsystem* prepares the environment for the compilation subsystem. It conveys the source programs to appropriate computation nodes so that they can be compiled using a proper compiler and linked with architecture-specific versions of the runtime libraries. This is especially important for heterogeneous systems or for nodes that do not share the same file system. The coordination subsystem transfers executable files directly if both nodes have the same architecture.

The coordination subsystem is also responsible for the operation of the WWVM as a dataflow machine. It constructs a dataflow graph by analyzing the task and dependency description information supplied by the user. The language used in writing the task's code, the types of message-passing libraries used (if any), and machines and domains that will be used to launch this task are all derived from this task description information.

Furthermore, when a complex metaproblem is to be solved, the coordination subsystem maps the individual tasks of the metaproblem onto different nodes of WWVM and manages overall coordination and synchronization of the tasks executed on each domain. It is responsible for starting up tasks and transferring data and status information between the source and destination nodes.

2.3.3 WWVM Virtual Machine Layer

The WWVM can be configured either as a dataflow machine or as a message-passing machine. This configuration is reflected in the way that the virtual machine layer connects the underlying computational resources [vonL 97]:

The *tightly coupled model* makes use of message-passing technologies for enabling fast communication of data and messages between processing nodes. The targeted programs are usually of medium grain. The *loosely coupled model* combines the use of several supercomputers and workstation clusters. Message or data exchange among the processing components of this computer is done with the help of files. The targeted programs are usually of coarse grain, and each subtask of the main problem is named as a *task (job)* with different processing requirements.

In the dataflow mode, message exchanges are usually infrequent, and message sizes are large. The processing time of tasks overrides the inter-task message passing time.

2.3.3.1 Dataflow Model: Loosely Coupled Computational Resources

The key point of the dataflow model is that all inputs must be present in order for a task to be executed. Within the dataflow model of computing [Sharp 85, Papa 91], there are two basic approaches to evaluation:

- Demand-driven approach (pull-model) in which data is pulled towards the sink or output stage as it is needed.
- Data-driven approach (push model) in which the arrival of data pushes information towards the output stage.

```
<TASK = T1 ;
 EXECUTABLE = task1.exe ; SOURCE =GAUSS.hpf :: HPF ;
 ARCHITECTURES = SUN4 , ALPHA ;
 INPUT = matrix.dat :: task1.in''; OUTPUT= "reverse.dat::task1.out1'', stdout::task1.out2 >
```

Figure 2 7. Description of dataflow tasks.

The WWVM uses a combination of the demand-driven approach and the data-driven approach in its dataflow mode of operation (Figure 2-6). A dataflow application needs to be divided into separate tasks with each *task* an independent computation entity (i.e. subproblem) that reads its inputs from a data file (either from the user through the Web-interface or from the output of another task) and writes its outputs to another data file or to the Web-based output console. Each task has its own separate source code, simplifying the process of porting individual tasks to different architectures.

WWVM uses a coordination language called *Task Dependency Description Language* that helps the programmer describe the main tasks of the problem and the interaction between these components to the coordination subsystem. This is similar to the language given in [BWD+ 93, TopH 96, THF+ 97]. The tasks are described as shown in Figure 2-7 using a task description editor that is embedded into the Web interface. The `TASK` keyword describes a task named `T1` that has an executable called `TASK1.EXE` on `SUN4` and `ALPHA` platforms. Heterogeneous systems, or systems that do not share the same file system, interpret this `ARCHITECTURES` statement and handle the transfer of the source codes to the appropriate processing nodes before compilation or execution. The executable codes between compatible machines can be directly transferred between servers. From the above description, the WWVM can identify that the source file, `GAUSS.HPF`, is written in HPF. In order to execute it, a file called `TASK1.IN` should be ready, and that saves its output into `TASK1.OUT1` and `TASK1.OUT2` once it is executed.

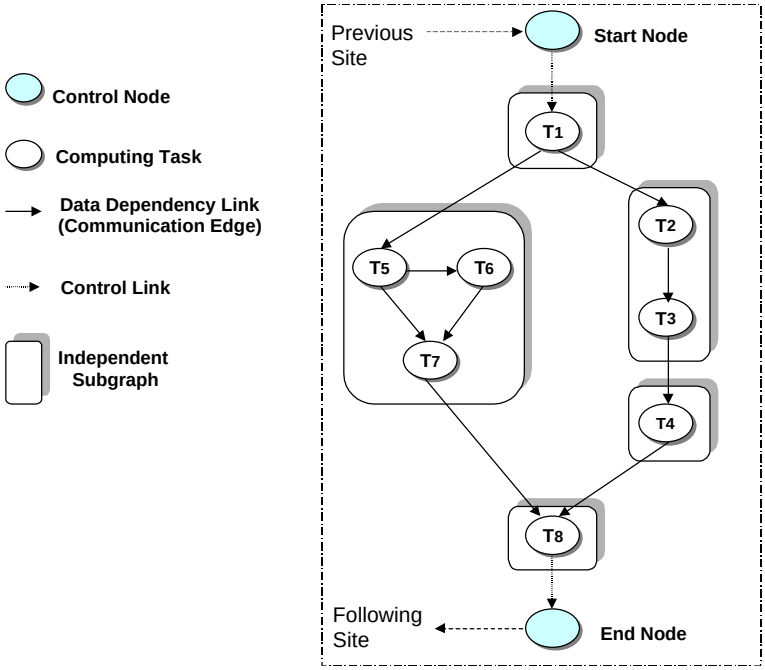

Figure 2 8. Definition of a problem and its task flow graph.

All tasks in a partitioned independent subgraph can be executed in parallel. Closely correlated tasks with similar platform requirements are brought together and can be launched on the same computation node. For such tasks the inputs from other nodes are read in at once before beginning to execute them, and all of their outputs are saved back into the output files when execution is finished. Therefore, most of the time they can use faster means of data- and message-passing mechanisms than the tasks that were scheduled on different nodes.

Once the task and task group descriptions are parsed and analyzed, an acyclic directed graph, the *Task Dependency Graph*, similar to the one in Figure 2-8 is constructed. The edges in the graph represent the precedence relationships between the nodes. These edges do not illustrate each individual input-output relationship between two nodes, since adding all of the communication edges would make the graph unreadable. This tree, augmented with platform

specifications, will later be distributed to all the WWVM servers that will participate in the execution of the problem. The local WWVM servers launch a task, once all of its inputs are ready after being notified by the WWVM.

Visual Programming Tools

At this point it is appropriate to mention a few of the graphical programming tools that is able define a program using a visual editor. The Heterogeneous Network Computing Environment (HeNCE) [BDG+ 92, BDG+93] is an automated tool for the development of heterogeneous applications. It was developed at Oak Ridge National Laboratories and built upon the PVM message-passing libraries. HeNCE provides a graphical interface called *htool* for graphical performance monitoring and for creating applications by supplying a set of C or Fortran function calls and a data-dependency graph. It also provides source-file distribution and compilation on remote hosts connected by PVM. It lacks the virtual management facilities of XPVM. CODE [BAS 85, NewB 92] is a visual tool that can provide architecture-independent parallel programs from any sequential program with the help of the user, who needs to specify the dataflow among serial parts. A parallel program is displayed as a directed graph where nodes represent sequential programs and arcs illustrate the flow of data between those nodes. The Abstract Visualization System (AVS) [Upson 93, AVS 92] helps to generate a static dataflow network of autonomous processes by a visual editor. A global control manager coordinates the passing of data from one process's output ports to another process's input ports by using interprocess communication or a remote procedure call mechanism. A module's operation is fired as soon as all the required inputs are presented at its input ports.

Implementation of the Dataflow Computing Model

Implementation of the dataflow computing model in the WWVM can be discussed on two levels: within a domain controlled by a single WWVM server, and among multiple domains. Completion of a task fires its successors, but they themselves pull the data to be used. We also assume that, within a domain, all platforms are using a shared file system. If one is not available, then several servers need to be started, one per file system.

Within a single domain the WWVM supports the dataflow computing model by using a mechanism very similar to a UNIX process called *fork* and *join*. The WWVM server employs a CGI module that analyzes the task-dependency graph, forks a new computation process to perform a task whose inputs are ready, and notifies the responsible server when the process is completed.

On the other hand, when a problem is distributed among several domains, the WWVM servers running on those domains should coordinate with each other in order to complete the tasks in an orderly manner. They use HTTP protocol for passing control messages. All the servers are distributed to the complete task-dependency graph and the code of the tasks for which they are responsible. The target WWVM compute servers fetch the inputs for the tasks assigned to them and notify the WWVM master coordinator selected to coordinate the computation of a specific problem when they are done.

Control flow is coordinated in a distributed manner. For performance purposes, no specific WWVM server is preassigned as a master coordinator responsible for coordinating the dependency relationships among different machines. The selection of the master coordinator is explained by the following example. Assume that a task such as the one given in Figure 2-8 is waiting for several other tasks that are presently executing on different domains. The WWVM server that is the closest common parent in the task dependency graph (e.g., T1) takes the

responsibility of collecting the status information of the tasks in subtrees. Once all prior tasks are completed, the dependent task is started. This kind of operation distributes the control mechanism to many servers and prevents some servers from becoming hot spots.

Pulling the data from appropriate tasks is done in a distributed manner. Each task gets its own data file from the supplying task's server location itself, either by using a customized form of `URL_GET` if it is on a remote server, or local copy or symbolic link mechanisms if it is on a local file system. It is assumed that the data is passed to other nodes on different machines by writing it to directories that can be accessed by WWW servers and reading it using Web-based access mechanisms on the other sites.

2.3.3.2 Message-Passing Model: Tightly Coupled Computational Resources

Besides the dataflow model, WWVM can also support the message-passing model by tightly coupling the computational resources. WWVM's virtual machine abstraction was built by using the PVM system's communication daemons. This virtual machine consists of the PVM daemons that reside on different nodes of the WWVM system. The PVM daemon structure was slightly modified to fit into the WWVM framework without affecting the interface seen from the outside. This gave the WWVM the ability to use existing codes written with PVM without making changes in them. This section will first give a brief overview of the PVM system and then discuss the capabilities of the virtual machine layer and the implementation details.

The Rationale Behind Using PVM

The Parallel Virtual Machine (PVM) is a software system that permits a heterogeneous collection of networked computing systems to be used as a single coherent, flexible, and concurrent computational resource [Sund 90, Sund 92]. PVM was adapted to the WWVM

environment instead of writing a communication layer software in order to eliminate the long initiation times and uncertainty associated with starting from scratch. As will be seen in Chapter 4, WWVM supports MPI and other message-passing paradigms such as Express and TCGMSG on top of its original PVM layer.

The choice of PVM over other message-passing libraries was not an arbitrary one – it was chosen for a number of reasons. First, PVM is a *de facto* message-passing parallel processing system that has gained widespread acceptance. It has been adopted by hardware vendors such as Cray and IBM, and efficiently implemented on specific hardware platforms. PVM was the most widespread message-passing library at the time this thesis work was conducted. Although it was expected that MPI would quickly become a message-passing standard, there was not a single, fully working MPI implementation available. In contrast, a public domain, full version of PVM implementation was readily available. In addition, the architectural details and implementation of PVM were well documented in a book (see [GBD+ 93]) that made it easy to understand and modify the PVM code.

Second, PVM supports dynamic processes and allows both SPMD and MPMD models of task parallelism. MPI and many other message-passing systems support only the SPMD mode of parallelism, where the number of processes is constant from the beginning of an application to the end. A third reason for choosing PVM was its unique process-management facilities. The execution environment may quickly change in the WWVM due to its Web-based processing components. The number, location, and capability of available machines and network cannot be predicted ahead of time. Worse, a program's execution environment may vary for different invocations. WWVM must deal with the dynamics of such a rapidly changing environment. The use of PVM allows programs to be developed for a uniform and predictable virtual machine. The dynamics of the underlying architecture is dealt with at runtime system level by exploiting the

PVM's dynamic process capability. New machines can be added to the current configuration, and PVM gracefully adapts to the new configuration. The last two properties are important in a dynamic environment based on Web and networked computing resources. Servers or individual machines may become available at any time or disappear abruptly. The WWVM needed to have explicit control over these configuration changes and be able to adjust gracefully to the changes in machine configuration.

Use of PVM also adds to the scalability property of the WWVM. PVM relies on UDP (User Diagram Protocol) for the underlying communication protocol between different daemons. Although UDP does not guarantee delivery of information as TCP (Transport Communication Protocol) does, this property is implemented internally by PVM. Using UDP instead of TCP affords three advantages: UDP consumes less system resources (file ids, etc.) and less Internet bandwidth, and it is faster than TCP. Since the number of hosts that may join into the WWVM configuration on the Internet is practically limitless, using UDP as the underlying communication protocol in WWVM is clearly more suitable than using TCP.

There are many direct advantages of using PVM daemons, some of which follow:

- *Fault Detection and Recovery*. The PVM's fault tolerance mechanisms ensure that the virtual machine will never run in a partitioned state, and as long as the master *pvmd* does not crash the virtual machine stays alive. A host failure is detected if an acknowledgment for a request to that host's PVM daemon is not returned. If a slave daemon cannot reach its master, it shuts down gracefully. When a host failure is detected, it is deleted from host tables, and all pending or future requests to that host return an error. Furthermore, even if application programs crash, *pvmd*s continue to run in order to aid debugging.

- *Data Coercion*. Messages are converted to a standard network format using XDR [XDR 87] before sending, and are converted back to the new local format after receiving when data is to be communicated between two hosts having incompatible data formats or binary data representation.

- *Reliable Message Delivery*. The PVM provides a buffering mechanism for holding messages. It delivers them in a reliable fashion and preserves their order.
- *Portability and Scalability.* The PVM has been ported to many different Unix platforms and runs under many different operating systems. Installing the PVM does not require any special system privileges. Most important, the PVM's decentralized and localized host and task management provides the capability of adding hundreds of hosts and running thousands of tasks.[1]

All these factors make the PVM very suitable for use in a Web-based metacomputing system. One general disadvantage to using PVM is its low performance as compared to MPI and others. However, this does not constitute a problem in the context of WWVM, since the targeted problems are computation-intensive.

PVM System Architecture

The PVM system includes system-level daemons, called *pvmd*s, which reside on each host in the heterogeneous computing system, and a simple but complete library of *interface functions*. The *pvmd*s provide services to both local and remote processes on other platforms in the heterogeneous computing system. They coordinate communication by serving as a message router and controller, spawn processes on the same machine, do authentication, and detect and

[1] In the current version of PVM, up to 4095 (i.e., $2^{12} – 1$) hosts, where 12 is the length of the host id are allowed.

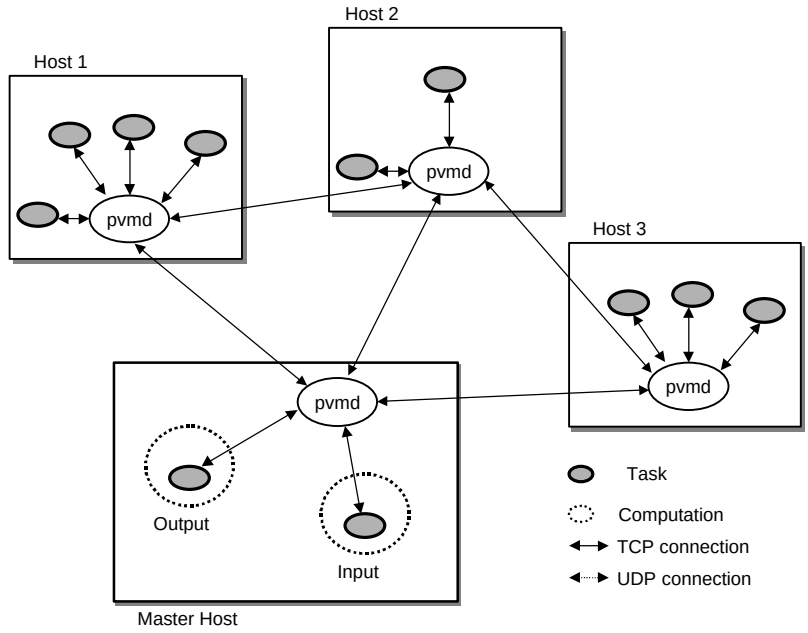

Figure 2 9. Logical view of a typical message-passing application on the WWVM.

report host failures. Together, the entire collection of *pvmd*s forms a virtual machine by enabling the heterogeneous computing system to be viewed as a single computer.

The basic unit of computing in the PVM is called a *task* and is analogous to a Unix process. The tasks can communicate by passing messages to other tasks through calls to PVM library functions. Library functions run in the same address space with the user's application.

Tasks that cooperate either through communication or synchronization are organized into groups called *computations*. The PVM supports direct communication, broadcast, and barriers within a computation.

Figure 2-9 shows a logical view of a typical application. An application generally starts with an *input and partitioning task* that controls how the problem is going to be solved. This task specifies how other tasks cooperate to solve a problem and creates several computations. Tasks

within each computation share data and communicate with each other. To communicate with a particular task, a task sends a message to its *pvmd,* which in turn forwards it to the *pvmd* on the destination host, which then passes it to the appropriate task. The application also has a dedicated task to handle output and user display. The other tasks in the application forward their output to this task for display. Their output is delivered to the *pvmd* on the master host through their local *pvmd*. In the WWVM the task responsible for output and user display passes all incoming items to the WWVM interface layer routines. These routines format the output in a suitable way in order to display it to the users via the Web-based interface.

Configuration of the Virtual Machine

Adding new computation nodes or deleting existent ones can change the configuration of the WWVM. Adding a new node is achieved by starting a communication daemon on the new machine. Once a daemon is started on a node it stays alive until that machine is removed from the machine configuration. A *pvmd* configures itself as a master or slave, depending on its command line arguments. The first pvmd started is designated as the *master*, and only the master can start and add new *slave* pvmds to the configuration or delete them. Reconfiguration requests originating on a slave host are forwarded to the master. Other than the configuration process, master and slave pvmds are equal. A *host table* describes the configuration of the virtual machine by keeping the name, address, and communication status of each node. An initial host table is compiled and maintained by the master daemon and is distributed to all other daemons at start-up time. It is updated in every reconfiguration and kept synchronized across all *pvmd*s.

Starting a slave *pvmd* requires initiating a process on a new host, and enough configuration information is passed to it for it to be added as a peer process. To assume the task of starting a new slave, the master *pvmd* delegates its authority to its shadow copy, and continues to wait for

messages from others. The shadow *pvmd* never talks to other *pvmd*s or tasks, but only communicates to its master via the normal *pvmd-pvmd* interface. The slave *pvmd* gets its parameters from the shadow *pvmd* via the command line and configuration messages. It creates and binds sockets to talk to other tasks and other *pvmd*s, and it opens an error log file /tmp/pvml.*uid*,where *uid* is the user identifier or id of the WWVM account owner.

The shadow pvmd uses `rsh`, `rexec` or manual startup to start each *pvmd*, pass its parameters, and get a line of configuration data back. Both `rsh` and `rexec` require a valid user account and password on each new host. The user id and password should be typed in the middle of the process. If the master machine is made to be trustworthy by the destination machine by the system administrator or by creating a `.rhost` file on the destination host, then this requirement can be released from the `rsh`. Similarly, if the user places his password in a `.netrc` file, then `rexec` skips the user authentication. However, these solutions hinder the security of local systems and make them more vulnerable to chain attacks. If a user's password is broken, the intruder may easily access his other accounts. Another disadvantage of using `rsh` or `rexec` is that when there is a single host that starts remote shells executing on other hosts, the number of open files quickly exceeds the limits of the operating system. Furthermore, the simultaneous execution of a large number of copies of the same executable file may overload the file server, causing multiple lost requests.

Using `rlogin` or `telnet` with a `chat` script may be another alternative for starting communication daemons on remote hosts. However, although this allows the accessing of hosts even behind firewalls, it is very slow and even PVM implementers have to refuse this alternative, since it is hard to make the `chat` program work reliably. Instead of using `rlogin` or `telnet`, we connected to a WWVM Web server already running on the remote host and started the slave

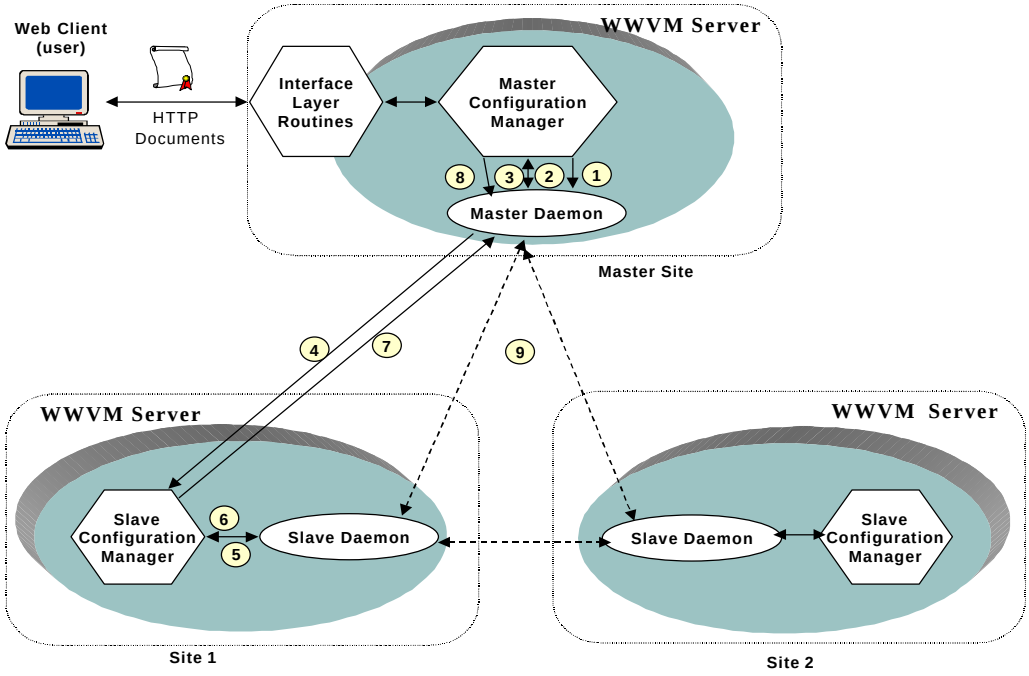

Figure 2 10. Steps for adding new hosts across the network to the virtual machine.

pvmd in a very fast and reliable fashion. A question may come to mind in regard to the possibility of connecting to an already running *pvmd* or *pvmd* server, or to the *inetd*. The reason we do not do that is that it would require the system administrator to install PVM on each host.

The quickly spreading Web technology has made it possible to run dedicated, secure Web servers responsible for WWVM configuration on each remote site. Servers extended with CGI functionality were used to start up local slave *pvmd*s, and they carry some of the configuration functions over these scripts. In our Web-based computing environment the user has no *real* accounts on each machine that would become a node of the WWVM. The WWVM takes care of accessing remote hosts and configuring the virtual machine in the way the user specifies. The user

may access the whole system by entering a WWVM password that opens doors to all other systems.

Handling WWVM Configuration Changes

The configuration subsystem is located at the master WWVM server site and gets the user's reconfiguration requests from the Web interface layer routines. It is used to add nodes to the machine, delete nodes from it, or to monitor the general configuration of the machine (see Figure 2-10). The configuration module passes the new configuration information to the Web interface layer routines after each configuration change. If the machine where the configuration manager is located participates as a slave node of the WWVM, then it accepts the incoming requests from the master site (or from other slave sites) and takes the necessary actions.

The reconfiguration operations are handled as follows:

- *Adding new nodes.* When a new node is going to be added to the configuration, the master site configuration module sends an ADD_NODE command to the local communication daemon and asks for the configuration parameters that are required to add new daemons (Steps 1-3 of Figure 2-10). Once this information is obtained, it opens an HTTP connection to the slave site's Web server (Step 4), automatically starts its slave site configuration module, and passes the master site configuration parameters to it. The slave site configuration module starts a local communication daemon and passes the incoming parameters to it (Step 5). In response, the daemon sends it its own configuration parameters (Step 6), which are forwarded to the master site as an HTML document (Step 7).

Upon receiving the document from the slave site, the master site configuration manager passes the information in the document to its local PVM daemon (Step 8), which completes the ADD_NODE operation.

- *Deleting nodes.* The master site configuration module instructs its local communication daemon to delete the specified node from the virtual machine. The daemon removes the address of the removed machine from its host table and broadcasts it to all other nodes.

- *Displaying the current machine configuration.* The master site communication daemon always has the current virtual machine configuration in its host table. It promptly conveys to the configuration module information such as the hosts, architecture types, and speeds of individual machines recorded in its host table.

These three types of operations are sufficient for configuration changes. In the next chapter other WWVM implementation details, which are mostly related to Web technologies, are discussed.