SchedSP: a Grid-based application service provider of scheduling solutions

G. Goulas*, †, P. Alefragis and E. Housos

Computer Systems Laboratory, Department of Electrical and Computer Engineering, University of Patras, GR-26500, Rio, Patras, Greece

SUMMARY

In this paper, SchedSP, a middleware framework for providing scheduling solutions as services over the Internet, is presented. Emphasis is given on creating a reusable framework that facilitates the development of specialized clients for the input, output and control interfaces of the various scheduling applications. SchedSP manages the task of preparing and running the required processes and allows the application interface developer to focus on the functionality and efficiency of the interface. The Internet-based scheduling applications created are competitive in all aspects with traditional locally executed applications. In this paper, detailed architecture and implementation details of the SchedSP framework prototype are presented. In addition, the methodology for creating specific case studies based on the SchedSP middleware framework is presented. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: application service provision; Grid computing; XML; scheduling

INTRODUCTION

Due to their combinatorial nature and the immense set of possibilities that need to be examined, real world scheduling applications usually require customized software and powerful computers. To tackle these problems, enterprises are required to undertake vast effort for the initial installation, maintenance and upgrading of various software and hardware components. This is only feasible for very large corporations with significant IT staff and computational resources. In order for these tools to be utilized by smaller enterprises, which might also be geographically distributed, a new computational model is needed. This requirement together with the fact that the Internet has expanded in quality and popularity led to the creation of the Application Service Provider (ASP) computational model in order to allow the distant user to have access to powerful applications without the need for heavily

*Correspondence to: G. Goulas, Computer Systems Laboratory, Department of Electrical and Computer Engineering, University of Patras, GR-26500, Rio, Patras, Greece.
†E-mail: goulas@ee.upatras.gr
investing in hardware, software and human resources. The ASP offers applications over the Internet for use on its computational resources using a flexible licensing, credit and/or payment scheme. One goal of the ASP approach is to narrow the software requirements on the client side to a Web browser. Such software can be considered as a universal feature, since it is installed automatically in all modern desktop operating systems.

In this paper, SchedSP, an Application Service Provision software framework that supports the development of scheduling solutions, is presented. Given the computational requirements of scheduling applications, SchedSP assumes that Grid resources provided by a Computational Infrastructure Service Provider (CISP) are used [1]. An important design goal of SchedSP is to allow favourable scaling on the number of concurrent users and allow for the efficient creation and support of new scheduling applications. The modular design of the SchedSP architecture provides support for the creation, storage and modification of input and output data on the server side. The application interfaces built on top of SchedSP are such that the input and output information can be visualized in a manner similar to the traditional use of a scheduling application on a local computer system. SchedSP also incorporates a set of generic scheduling related components in order to assist the application developers in the creation of their scheduling systems.

The initial motivation for SchedSP was to provide the scheduling applications, which have been developed over the years in the Computer Systems Laboratory of the University of Patras, as application services over the Internet. These applications cover problems from several scheduling domains including university timetabling [2], high-school timetabling [3–5], personnel scheduling [6], bus driver scheduling [7], and airline crew scheduling [8–11].

Several published papers discuss various methodologies for the organization and architecture of a service-provisioning framework. In Vinci [12], a service-oriented architecture for supporting the development of Web applications is proposed, using XML [13] document exchange, which is in several ways similar to the SchedSP philosophy. Middleware for Method Management presents a Web-based system for the sharing of statistical computing modules [14–16]. The use of XML for invocation of remote procedures has been proposed in several documents, reports and standards like the XML RPC [17] and the XML Web Services protocol suite [18] of SOAP [19], WSDL [20] and UDDI [21]. Products that utilize the XML Web Services specifications include Microsoft .NET framework [22], the Sun ONE [23] architecture and the IBM WebSphere platform [24].

The paper is organized as follows. Initially, a quick introduction to Application Service Provision, Grids and a typical scheduling application scenario is given. In the following section, detailed functional specifications of the SchedSP framework and its characteristics are presented, followed by an extensive discussion of the Service Provisioning Components based architecture of SchedSP. Finally, use case experience, conclusions and future work is presented.

APPLICATION SERVICE PROVIDERS

An ASP is an organization that offers applications as services over the Internet, usually on a monthly, yearly or at a per-usage payment structure model. The ASP computational model is often defined as ‘a form of selective outsourcing where a third-party organization rents generally available packaged software and related services’ [25, p. 197]. The evolution of the ASP model very soon proved that different application types have substantially different needs in terms of design and infrastructure.
Thus, specialized types of service providers emerged, collectively referred to as xSPs, where ‘x’ denotes the specialization of the provider. Such service providers include Database Service Providers, Management Service Providers, Security Service Providers, Storage Service Providers, CISP and others.

The quality of the service offered by an ASP to a customer is described in a contract named the Service-Level Agreement (SLA). This document defines several worst-case guarantees in terms of measurable quality aspects of the service offered, including availability of the applications and response times to specific user actions. Several other application-specific measures may be included in an SLA, including disk space availability when file or database storage is provided. The application availability is difficult to predict, since it depends on the networking and computing equipment reliability, the overall infrastructure, interconnection and redundancy strategy and the quality of the software in use. The response times to user actions on the other hand, which are directly relevant to the user vision of the efficiency of the service, are directly dependent on the available computational power and scalability of the service and the networking infrastructure of the ASP. The design of these two features should provide appropriate margins in order to satisfy a reasonable number of simultaneous users.

A particular service provider type that is of primary interest in our work is the CISP, whose task is to provide computational infrastructure as a service over the Internet. The computational infrastructure, with additional support from the appropriate Grid creation middleware, facilitates the execution of both standard sequential and parallel/distributed programs. Condor Resource Manager [26], Platform LSF [27] and Grid-computing platforms [28,29] can be used for the organization of CISP computational resources. In most cases, a CISP offers a remote procedure call interface, similar to the XML Web Services protocol suite, to its customers [18–21]. Such an interface enables an ASP to use the CISP resources in order to acquire additional computational power for the computational intensive parts of its applications. This idea is illustrated in Figure 1, where an ASP utilizes CISP computational resources besides its own.

The benefits of a CISP are mainly organizational and provide a reasonable mechanism for risk management and the reduction of initial investment for the ASP. Since the resources of the CISP...
are shared across different ASP organizations, all parties involved appear to benefit from such an arrangement. Of course, the SLAs governing the relationship between an ASP and a CISP should provide the ASP with the means to form reasonable SLAs with the end customers. Another potential scheme can occur when the ASP internally organizes its own resources as a CISP, since then it can also offer its computational infrastructure for use as a separate service.

A prototype CISP system, tightly linked to the Grid organization of its computational resources, has been developed at the Computer Systems Laboratory of the University of Patras. The system is called PLEIADES [1,30] and it enables the use of a set of Grid resources scheduled by the Condor Resource Manager [26,31,32] through an ‘XML protocol over HTTP messaging’ type interface. In addition to the messaging interface, there exists a Web-based user interface to PLEIADES that allows for the direct interaction of regular users to the system. The basic functionality of PLEIADES is called iNOW and allows for the formation of Internet-based computational clusters for the execution of a specific job. The services offered by PLEIADES are shown in Figure 2 and include job submission to a virtual cluster, monitoring, job control and a virtual file space service to be used for input and output purposes. PLEIADES also provides a Cross-compile service, which utilizes machines of different architectures and operating systems to create the corresponding executables.

AN INTRODUCTION TO SCHEDULING APPLICATIONS

The aim of a scheduling application is the assignment of a set of resources to a set of tasks in order to produce a feasible, fair and efficient schedule. Many types of scheduling applications exist but it is not our purpose to elaborate on all of these possibilities in this paper. An overview of scheduling algorithms can be found in [33]. All scheduling applications must satisfy a set of legality rules and regulations while attempting to locate the optimal assignment of resources to the various tasks. For example, in the timetabling problem, teachers, classrooms, class hours and student groups must be considered in order to produce a feasible schedule. In addition, temporal constraints, which impose sequencing between tasks, like an exercise hour should occur after the theory hour and during the same day for a specific course, also exist.

Scheduling applications that involve human resources are even more constrained with respect to the various rules and contracts that must be satisfied and are in daily use by various organizations,
corporations and institutions. The wide use of such scheduling applications stems mainly from the fact that there are significant benefits produced both in the quality of employee working patterns and on the minimization of the required number of working hours. Expensive optimization software with significant computational requirements are the basic components of scheduling applications, because scheduling problems are usually modelled as large-scale mixed-integer optimization problems. Even when considering the 0–1 set covering or set partitioning simplification of the problem, it still belongs to the NP-hard problem category. It is also a fact that usually such applications are not used every day. It is easy to assume that a small or medium-sized enterprise would not buy expensive equipment and software, although it is required in order to improve the performance of the enterprise. A possible solution in this business situation could involve the outsourcing of the scheduling service to an external enterprise, provided that the data are secure and the availability guaranteed. Such an approach would significantly reduce the overall cost, leverage the quality of the service offered and enterprises would not have to maintain complex and often under-utilized computing systems and applications.

Figure 3 presents a generic design pattern that scheduling application designers usually follow. Initially there exists a pre-processing phase in order to reduce the size of the problem using domain knowledge, previous solutions and simple logical inferences based on resource characteristics. The problem can usually be logically partitioned into subproblems based on some criteria, for example, the single airplane cockpit qualifications of the pilots lead to the creation of aircraft fleet based subproblems for the airline crew scheduling problem. These subproblems are solved using various algorithms, which in fact iteratively generate alternatives while attempting to improve the current subproblem solution. This process covers a broad range of problem-solving procedures from the highly mathematical linear and integer programming to the various heuristic and meta-heuristic methods. All methods require that there is a module that fully describes the rules and regulations that individual resource schedules should comply with, and acts as the main guide used in the process of discarding
infeasible scheduling alternatives. Another required module that always must exist provides the costing of the alternative scheduling options. The calculation of these costs is often complex and of a nonlinear nature. The optimization step selects the best feasible resource assignment pattern that satisfies global capacity, sequencing and/or availability constraints. Finally, a post-processing step combines sub-problem solutions and improves the global solution by applying local optimization strategies.

An important characteristic of scheduling applications is the use of more than one algorithm or method to solve a particular problem [34]. Usually there exists a strategy selector module that acts as the orchestra conductor whose job is to orchestrate the available solution methods, based on feedback information delivered by the solvers that has been utilized in the previous iterations. Based on our experience from various production-level scheduling applications, it is not possible to predict that a certain solution method would deliver the best result for all problem instances. Thus, scheduling vendors usually create a toolbox of components that when appropriately interconnected can create an application that can efficiently solve all problem instances. This approach allows the users to select the solution approach that best fits their problem.

THE SCHEDSP FRAMEWORK

The creation of applications that are offered over the Internet as services involves the design and implementation of common functionality for each particular application domain and some application-specific components. When these applications have a significant set of common characteristics, for example, a set of scheduling applications, development work can be reduced significantly with the use of a software framework. During the last decade, the Computer Systems Laboratory has participated in many scheduling application related research projects. The efficient execution of these applications required a specialized computational environment. Recent technological advances urged us to think about these applications and their use in a new perspective. Our vision is to provide them over the Internet for public usage, since many of them have been supported by different kinds of public funding and do provide solutions to useful scheduling problems. The framework that has been designed and implemented for this purpose is called SchedSP [35], standing for Scheduling Service Provider, in direct correspondence to the xSP naming conventions. The SchedSP system must satisfy all the availability, client side flexibility and scalability requirements of a typical ASP while creating additional support tools for the scheduling application domain. Having this motivation, pricing and billing issues have not been thoroughly considered for the initial design and implementation of the SchedSP prototype, although open design principles guarantee the means to incorporate them in the future.

The main requirement of the SchedSP framework is the ability to efficiently support the creation of Internet-based scheduling applications. The main scheduling application attribute is the need for a large and often special computational environment. It is also assumed that all applications are able to run in console mode and they can be controlled by another program using redirected standard inputs and outputs. As a framework for developers, SchedSP has to provide a consistent and coherent set of services in order to minimize the development effort of new software. The core services must follow a common interface pattern in order to facilitate their use. Another important factor is the efficiency of the system, which involves the throughput of the production of schedules but also the support for simultaneous usage of the rest of the supporting services. Scalability is another important factor, since it
can be a metric of how many simultaneous customers an ASP could serve. Availability and efficiency constraints are also considered. Security mechanisms that protect users and data in transit are also important design issues.

While the design of SchedSP was based on all of the above requirements, it mainly addressed a subset of them, as detailed in the next section. Some of them, like the complete security requirements, are only partially examined. A commercial deployment of SchedSP would also require pricing and billing support.

The first design choice was the use of a Grid-type infrastructure [28,29] for the execution of scheduling applications. This resource organization provides the necessary services for accessing a distributed high-capacity computational environment, either for parallel/distributed or traditional serial computations. In most scheduling situations the users require results within a reasonable time window. Due to practical reasons (working hours, seasonal needs, etc.) scheduling requests follow a non-uniform pattern, with spikes at the beginning of scheduling periods, like semester beginning for university and college timetabling. Thus it is essential to be able to utilize a Grid-type back-end processing system and also be able to utilize additional external computational resources whenever needed. For these reasons, SchedSP has been designed to utilize resources from the PLEIADES CISP environment. The SchedSP framework services are grouped into functional sets and implemented within components called Service Provisioning Components (SPCs). In the prototype, the components are accessed using the Microsoft DCOM environment. The selection of the appropriate component for each request is dynamic. This enables an SPC upgrade to be performed by changing the mapping between the particular service and the appropriate version of this SPC component. This feature allows for major upgrades without loss of service availability. In addition, the use of DCOM allows for the distribution of these components across a Local Area Network (LAN).

An important factor for the end users to be productive in the ASP computational model environment is that the user interfaces of the SchedSP-based scheduling applications should closely resemble in quality and ease of use a typical local scheduling application environment. With respect to the client software distribution scheme utilized in order for the end user to use a particular application, one of the following options could be used. The first approach is the installable client option, where the user downloads and locally installs a version of the client interface. A new version must be downloaded whenever changes have occurred. The second option involves the use of a thin client, where the user interface is downloaded in every use. The thin client approach is more appropriate for a service provider interface, whether it is a lightweight Web-based interface or a rich interface based on Java Applets, Macromedia Flash or Microsoft ActiveX components. The benefits of the thin client approach include the lowest possible maintenance costs for the client and the possibility for the user to access applications and files from any computer with Internet access.

Currently, the communication between SchedSP and its clients is based on ‘XML protocol over HTTP messaging’. Several XML schemas have been created for the definition and validation of these messages. XML as a design choice for all SchedSP communications presents several advantages but also certain drawbacks. XML documents are self-described for a human if the various element and attribute names are reasonable. Client developers were able to use SchedSP services just by examining the available DTDs and inline comments, together with a set of request/response pair examples. An XML document can be extensible without breaking applications based on previous schema definitions. This allows for continuous service upgrades without breaking the available clients. Furthermore, the usage of a proven widely used XML parser library guarantees the minimization of...
any errors in the process of parsing messages. A specialized parser of binary message would always outperform an XML parser and present a lower network load. The design choice for SchedSP was to trade off some efficiency with ease of use for the messaging mechanism. In addition, the time required to solve a scheduling problem is greater by many orders of magnitude than the time required to parse an XML message.

SCHEDSP ARCHITECTURE

Figure 4 shows the main service categories that SchedSP offers in order to satisfy the requirements described in the previous section. The SchedSP architecture elements have access to the system database that stores user and session information, the service mappings to components and SPC configuration settings. The services are implemented using discrete components SPCs. Each SPC implements a ‘doServe’ method, which performs the requested service. Details for each service together with its corresponding SPC are discussed and analyzed in the following section. An overview of the SchedSP architecture is shown in Figure 5. An important benefit of this architecture is that it separates the service provisioning from the tedious but critical tasks of user authentication and authorization, which are performed by the Executive component. When the Executive processes a request, which is encoded in XML, it categorizes it according to its Document Type Definition (DTD) or schema. Then, it queries the database to find the name of the corresponding SPC, and calls the ‘doServe’ method of the specific component. The ‘doServe’ method takes as arguments an XML document and a reference to a shared DCOM component called SchedSP_Settings. The XML document contains the request for service that has been validated by the Executive against its corresponding DTD. The SchedSP_Settings component encapsulates several settings, populated by the Executive component, like session characteristics and resource mappings that an SPC needs in order to properly perform the required functionality. For access control purposes, the Executive and the SPC components consult the Access Control List (ACL) component. SchedSP assumes that the execution of the scheduling applications will be performed on the PLEIADES Grid infrastructure and for this purpose specialized SPC components have been created. Assuming that an SPC upgrade is needed, the new version of the component is installed on the server and the service mapping is updated in the system database. As a result, the newly installed SPC automatically serves all new requests. It is important to maintain compatibility across the different versions of the same SPC, in terms of XML document structure and service functionality.

The communication protocol between the client software and SchedSP follows the three-layer protocol stack shown in Figure 6. The HTTP protocol is used for message transfer. The HTTP protocol is chosen because it follows the client–server model and usually is able to use any type of Internet access, including the Network-Address-Translation (NAT) mechanism, or pass through firewalls. The HTTP layer corresponds to the Gateway component, the SchedSP layer to the Executive component and the SchedSP Service Provisioning layer to the SPC components. The Service Provisioning layer messages are encoded in XML and for each SPC component the appropriate DTD must be defined. Figure 7 presents the interaction scenario involving a SchedSP Client, the Executive and the various SPC elements. For simplification and readability purposes, the Gateway role has been omitted from Figure 7. The SchedSP Client initiates the interaction with a user authentication request, similar to the message presented in the left part of Figure 8. If the user credentials are valid, the
Executive responds with a success message including a session identification tag. The user name and session identification tag pair form the user credentials, valid until either a logoff or session expiration occurs. These credentials are needed in order for SchedSP to accept SPC requests. The SPC requests are enveloped by the SchedSP Client in an XML document similar to the example presented in Figure 9. The Executive forwards the SPC request part of the XML document to the appropriate SPC and returns the SPC response to the SchedSP Client. A typical logoff message is presented in the right part of Figure 8.

**SCHEDSP PROTOTYPE**

The architecture of the implemented prototype is shown in Figure 10. The following sections describe the details of the components of this prototype.
Figure 6. SchedSP communication layers.

Figure 7. SchedSP communication interaction.

Figure 8. Authentication (left) and corresponding logoff message (right).
<?xml version="1.0">
<!DOCTYPE schedsp ...>
<schedsp>
  <user value="goulas" />
  <session value="asdasfa2sa3" />
  <request>
    SPC Request
  </request>
</schedsp>

Figure 9. Structure of a SchedSP message containing an SPC request.

Figure 10. SchedSP prototype architecture.

Gateway

The Gateway component is responsible for message transportation over the Internet. It implements part of the HTTP layer of the SchedSP communication stack, shown in Figure 6. In this component, several common Web server security techniques are applied. It is implemented as a Microsoft ActiveServer Pages script [36]. When a request arrives, it parses the request, unpacks the contents and executes the Executive component.
Executive

The Executive component is the central manager of the SchedSP system. From a communications point of view, it implements the SchedSP layer of the communication stack, shown in Figure 6. The requests for the Executive component are encoded in XML and conform to the DTD described in the ‘schedsp.dtd’ (Figure 11). The available request possibilities include user authentication, session termination and SPC requests. The Executive follows the interaction scenario presented in the previous section. The Executive queries the ACL component if the specific user has the appropriate authorization to use a service provided by an SPC.

Access Control List

The ACL component provides an access control mechanism to limit user access to the various resources. The model followed is the role-based access control [37], where rights are described for a set of roles, while in parallel, certain rules might apply to individual users. Both file and computational resources are defined hierarchically. When no individual access rules are defined for a specific resource, the inherited access rules are applied. The application of access rules is realized in a hierarchical manner, so that explicit denials of access at a higher level have precedence over allowance at a lower level. The ACL component can be accessed either using a DCOM interface for SchedSP internal usage or through the common API of the SPC components, so that SchedSP clients are able to perform access control queries or set access rights for their resources.

Execute and monitor

The Execute SPC is used to submit scheduling applications for execution to the PLEIADES CISP Grid resources. For each run, the Execute SPC stores all required files in a new working directory in order to maintain a direct association of the input and the output files. The parameters of the Execute SPC are the scheduling application, additional command line arguments, input/output files names and input, output and error channel redirection files. The execute component utilizes the PLEIADES filesystem service to store these files on the PLEIADES virtual file space. To begin execution, the Execute SPC retrieves the scheduling application execution profile from the database and utilizes the PLEIADES...
services to submit the program for execution. The PLEIADES application execution profiles include information like the required operating system and platform. In case of successful submission, the Execute component returns the identifier it received from the PLEIADES system, in order to be used by the SchedSP client for job monitoring purposes. The Execute component utilizes the ACL SPC to determine the user permission on the execution of the program.

The Monitor SPC is used to monitor jobs submitted for execution at the PLEIADES system. The component returns a set of details for every job matching the provided parameters, which are optional and may include the user name, the job identifier and the job status. Since the HTTP protocol follows a pure connectionless client–server model, it is impossible to directly notify the SchedSP client for the completion of a submitted job. To overcome this, the option to receive an e-mail upon job completion has been implemented. Although this is not a complete remedy, it is an acceptable alternative since the completion of a scheduling application requires a sizeable amount of time. Another significant issue is the feedback information about the progress of the scheduling algorithms received by SchedSP in order to inform the user. Progress information is maintained by SchedSP and can be made available to the user. Progress information is collected through the Scheduling Coordinator executable that is described below.

**Filespace**

The Filespace component provides a virtual storage area for input and output files needed by the scheduling applications. The Filespace queries the ACL component every time a file operation is carried out. It supports hierarchical organization of files into directories and implements a set of typical manipulations on files and directories including creation, removal, copy, rename and move. The provided services are presented in Table I. The files and directories visible through this service are real files on a special directory of the SchedSP server. The SchedSP Settings component translates client application domain path names to server domain ones. There exist four top-level directories that are visible to the user, which are organized as follows: ‘HOME’, which maps to the user personal directory, ‘COMMON’ used for file sharing between users; the ‘EXECUTABLES’ directory where the scheduling applications reside; and the ‘XSLT’ directory which is a repository for the XSLT filters, which describe transformations between XML document types.

**XMLTrans**

The XMLTrans component performs a set of transformations (Table II) to XML files stored in the SchedSP file storage area. The SchedSP system encourages the use of XML documents for data storage. This decision was based on the inherent advantages of XML for the development of data exchanges. The first XML transformation for XMLTrans is based on the XSLT specification [38], either using XSLT filters stored in the special ‘XSLT’ directory or user-provided XSLT filters. The second is a custom transformation based on the XPath specification [39] and includes addition, removal and exchange of any element, attribute or subtree of an XML file stored in SchedSP. The designated node for modifications in a request is addressed using the XPath standard. In case of replacement, the new node is also contained in the request. This service is very useful since users often make minor updates to their pre-existing files.
Table I. File space operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkdir</td>
<td>Create a directory</td>
</tr>
<tr>
<td>rmdir</td>
<td>Remove a directory</td>
</tr>
<tr>
<td>rmtree</td>
<td>Remove a directory tree</td>
</tr>
<tr>
<td>mvdir</td>
<td>Move/rename a directory</td>
</tr>
<tr>
<td>cpdir</td>
<td>Copy a directory</td>
</tr>
<tr>
<td>touchfile</td>
<td>Create an empty file</td>
</tr>
<tr>
<td>rmfile</td>
<td>Remove a file</td>
</tr>
<tr>
<td>mvfile</td>
<td>Move/rename a file</td>
</tr>
<tr>
<td>cpfile</td>
<td>Copy a file</td>
</tr>
<tr>
<td>dir</td>
<td>Show directory contents</td>
</tr>
<tr>
<td>view</td>
<td>Show file contents</td>
</tr>
<tr>
<td>store</td>
<td>Store data in a file (replace)</td>
</tr>
<tr>
<td>validatexml</td>
<td>Check if an XML file is valid/well formed</td>
</tr>
</tbody>
</table>

Table II. XMLTrans operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslt</td>
<td>XSLT-based transformations</td>
</tr>
<tr>
<td>xmlactions</td>
<td>Actions for XML files, as defined below. XMLTrans input can describe a sequence of such actions</td>
</tr>
<tr>
<td>addElement</td>
<td>Add an element in the XML tree</td>
</tr>
<tr>
<td>remElement</td>
<td>Remove an element</td>
</tr>
<tr>
<td>changeElement</td>
<td>Exchange an element with the one provided</td>
</tr>
<tr>
<td>addAttribute</td>
<td>Add an attribute</td>
</tr>
<tr>
<td>remAttribute</td>
<td>Remove an attribute</td>
</tr>
<tr>
<td>changeAttribute</td>
<td>Exchange an attribute with the one provided</td>
</tr>
<tr>
<td>addSubtree</td>
<td>Add an XML subtree (starting with an element)</td>
</tr>
<tr>
<td>remSubtree</td>
<td>Remove an XML subtree</td>
</tr>
<tr>
<td>changeSubtree</td>
<td>Exchange a specific subtree with the one provided</td>
</tr>
</tbody>
</table>

MultiServe

The MultiServe component allows for automatic execution of a set of service requests in a single request message to SchedSP. A MultiServe request contains a sequence of Service Provisioning layer messages and some flow directives in the form of XML elements. These directives allow for case-specific execution of the SPC services based on the responses provided by previous SPC requests. Therefore, this service is in effect performing batch-type actions and thus eliminates the message round-trip delay among a set of separate service requests. The MultiServe implementation needs access to the Executive selection mechanism. For this reason, the selection mechanism is implemented as a...
separate COM component that is usable by the MultiServe and Executive components. In case of an error condition, a rollback action can be defined in order to return the system to a previous state. Finally, an aggregation of all SPC responses is returned to the client together with an indication of success or failure.

The main available constructs of MultiServe input are SPC calls, variable declarations, loops, conditional statements, ‘on Error’ and rollback actions. SPC calls enable MultiServe to send XML requests to an SPC using MultiServe variables as XML input and output. The MultiServe variables are named text or XML placeholders. The XPath query mechanism is used to define the exact value assigned to a specific variable from an XML document. In case an XPath query returns multiple results during a variable assignment, the operation fails and any associated ‘on error’ or rollback actions are applied. Loops can repeat a set of instructions for a specific number of times or for all the list members of an XML nodes collection returned by an XPath query. A conditional statement is able to change the instruction flow based on the existence of an XML node, the comparison of variables, etc. ‘On Error’ statements define simple error trapping conditions and error recovering procedures. In case an error occurs when an error trapping condition has not been defined, a rollback action is called. Rollback should undo previous actions and return to a valid previous state. Rollback is not automatically generated and the client developer is responsible for the definition of the appropriate and consistent rollback actions for the particular MultiServe script.

**SchedScripter**

The SchedScripter SPC allows for execution of a solution strategy for a particular scheduling domain in a single request message to SchedSP. It introduces a simple scripting language in order to describe high-level solution policies that the system should follow in order to solve problems of a specific scheduling domain using the available scheduling modules. Using domain-specific profile information and user parameters as input, the SchedScripter SPC generates an XML document that is used by the Scheduling Coordinator executable. Profile information includes flow control conditions, required scheduling executables and libraries and required intermediate results and solution targets.

**Scheduling Coordinator**

The Scheduling Coordinator executable implements an XML parser that uses XML documents generated by the SchedScripter SPC to apply solution strategies. The Execute SPC submits the Scheduling Coordinator to the PLEIADES CISP along with the XML document that describes the solution strategy, other required executables and input files. Based on the directives, depicted by the XML document, decisions on the selection, the order and the parameters of the scheduling applications that should be used in order to solve the particular problem are taken. The user also describes possible data translation actions that have to be taken to continue the solution process. The decisions are based on the solution progress and feedback information of the previously selected policies and algorithms. This approach requires that the available scheduling applications can also be used as autonomous software modules that given certain input and a set of parameters, provide output every time they are called. This is a restriction for older scheduling applications, where actions and algorithms were fully defined and organized in the source code of the solvers. In this situation it is not
possible to dynamically change the order of execution and thus only the data management part of the scheduling coordinator is in effect. The final output data are returned to SchedSP.

**Legality and Attribute Evaluator**

The Legality and Attribute Evaluator (LAttE) component allows for the expression and evaluation of the rules and regulations governing the scheduled tasks assigned to specific resources. Scheduling rules implement the business logic that all scheduling solutions must follow. Since scheduling rules are continuously changing due to the dynamic nature of the competitive business environment, there is a need for a high-level domain-specific language in order to express and manage these rules.

Domain-specific component instances that implement these rules in SchedSP are generated using a high-level domain-specific language [40]. To enable common representation, the development of a generic legality system was required. Applying domain analysis to the scheduling problem domain, an object meta-model was created [41], the main part of which is depicted in Figure 12. The basic building elements of the meta-model are activities, properties, time windows, and rules. At this time it was apparent that the evaluation of activity properties was at the core of the LC development. These attributes and their distinct entity appearance in the meta-model provide the infrastructure for their individual calculation which is responsible for the attribute evaluation flavour of the produced system.

The attributes that are calculated to evaluate the rules are also used to calculate cost and schedule characteristics, information that can be actively used in the generation and optimization phases of the solution process. Using the scheduling rule language the user is able to change not only the parameters but also the structure of the rules. The important benefit of this approach is in the ability to perform

![Figure 12. Legality and attribute evaluator meta-model.](image-url)
what-if scenarios and test for rule extensions and additions without changing the application programs. In addition, it allows the dynamic modification of the rules without disturbing and risking the integrity of the scheduling applications. The component follows on the distributed system design that was initially used in production by a major European airline [42].

The main adaptation to the LAttE component, in order to satisfy SchedSP design requirements, involved the addition of an XML document parsing interface. The LAttE component can be accessed in two different ways. Similar to the ACL SPC component, a set of methods is exposed as a COM interface that the various user client applications can utilize in order to determine the legality of user actions in the planning and management of scheduling solutions. This is very common in scheduling applications as the users usually manually adapt the solutions provided by the scheduling algorithms in order to satisfy discrepancies due to resource unavailability, demand changes, etc. The same LAttE module can be accessed through an API by the various scheduling subsystems in order to evaluate the feasibility of alternative scheduling sequences. In this manner, the component manages to provide a coherent module for all related scheduling systems, providing a single system for maintenance and support and a common language to express all required scheduling rules.

**SchedSP Client Kit**

Although the SchedSP framework exposes an easy to use XML-based client interface, the composition of the various XML-based requests is a common functionality for all SchedSP clients. SchedSP client developers can use the provided Client Kit in order to generate easily the various XML documents. While the XML over HTTP rationale outlines a cross-platform and language-neutral solution, the creation of a Client Kit cannot further inherit this feature. Thus, the SchedSP Client Kit is provided in several different implementations with identical functionality in order to support a variety of programming languages and platforms. The provided implementations are an ActiveX component, a Java package and a Microsoft .NET assembly. Figure 13 contains a simplified class diagram of the Client SDK. The SchedSPClient class utilizes the available HTTP transfer mechanism and provides the SchedSP layer of the communication, which contains user authentication, session management and SPC request transfer. The ‘SPC clients’ classes derive from the SchedSPClient and contain member methods that directly match to the various operations provided by the corresponding SPC. These members compose the XML-based SPC requests and utilize the SchedSPClient mechanisms to transfer the requests to SchedSP. The client application creates a SchedSPClient instance from the Client Kit and it calls the appropriate methods to authenticate the user. Following authentication, any SPC Client object can call SPC-specific methods using the created SchedSPClient instance. All attributes and methods shown in the SchedSPClient class in Figure 13 are final, so that the derived classes cannot redefine them.

**SCHEDSP USE EXPERIENCE**

Several applications based on the SchedSP framework have been created. This was done in part to verify the ability of different end user developers with various degrees of scheduling and programming experience to design and implement scheduling clients. The first experiments involved the creation of ‘SchedSP Explorer’, ‘MediSched’ [43], ‘KTEL.Sched’ [44] and ‘SchoolSched’ applications.
The ‘SchedSP Explorer’ application is an ActiveX component to explore the SchedSP file space and manipulate files and directories, similar to other file managers. ‘MediSched’ is composed of an ActiveX-based client that acts as the end user interface for the nursing personnel scheduling application described in [6], which is also available to SchedSP. Similarly to ‘MediSched’, ‘KTELSched’ is also the combination of an ActiveX-based client and the appropriate scheduling solver [7], for the combined bus and driver scheduling problem. The scheduling solver is in use by various bus companies around Greece. The ‘SchoolSched’ produces timetables for Greek high schools and is composed of a Java applet for the client–user interface and the scheduler described in [4].

The creation of an Internet-based scheduling application using SchedSP, although it appears cumbersome, is indeed a reasonable exercise and allows for the creation of well structured and robust systems. While the use of SchedSP appeared to add effort compared with other schemes of creating an Internet-based application, the conclusion of these use cases was that SchedSP was indeed a very easy and efficient system to work with. All developers said that the services’ behaviour was always as expected; the various names and comments within every DTD made the usage of each SPC and its corresponding services clear. To further facilitate creation of application clients the SchedSP Client Kit has been created. A ‘SchedSP by example’ document, where a subset of the debugging test inputs was presented, significantly helps in the development effort.

Since all the scheduling applications used as back-end solvers need a set of input files for the problem definition and produce a number of output files containing the proposed solution, the directory structure of the execute module was indeed very useful. The output and error streams of these applications contain useful information and errors, and need also to be captured. The use of XML for all the files was a design choice for all applications, since the XMLTrans SPC provided extra benefits for the usage of XML files, but the scheduling applications provide their output in a custom format. To overcome this
limitation, custom wrapper programs encapsulated the scheduling applications to utilize XML for input and output. The application profiles, which are stored in the database and used by the Execute SPC, contain instructions to also include the wrapper in the files package submitted on the PLEIADES CISP, and to use this as the entry point instead of the real application. When any of the scheduling programs is submitted, the application interface is monitoring the job at certain intervals to determine its execution status. The polling from the interface is necessary since the client–server model is applied to the interaction between the user interfaces and SchedSP. A similar problem with the client–server model has arisen in the XML Web Services domain and one proposed solution is the ‘publish–subscribe’ type protocol named WS-Eventing [45]. In the SchedSP situation, the polling interval, typically selected from tens of seconds to some minutes, is short compared with the time to completion of the various scheduling problems. Another problem that had to be solved involved the inability to predict the completion time of each scheduling run. The execution time is not a linear function of the input size and the problem structure, while the temporal characteristics of the availability of the PLEIADES resources as well as the computational capabilities of the machine chosen may vary significantly among the various runs. When the scheduling process finishes a notification email is forwarded to the user in order to minimize busy waiting by the users.

**SchedSP Explorer**

The ‘SchedSP Explorer’ is an ActiveX component for file and directory manipulation within the SchedSP File Space. It actually presents all the functionality exposed by the FileSpace SPC. Besides using it for server side storage, it also served as the basic code example for the rest of the applications.

**MediSched**

The ‘MediSched’ application was created as a diploma thesis project and its purpose was to create working schedules for the nursing personnel of a hospital department. A hybrid algorithm, using an approximate Integer Linear Programming (ILP) model with Tabu search strategies, has been researched and developed at the Computer Systems Laboratory by Dr Valouxis, for the creation of nurse scheduling solutions of local medical institutions [6]. An integrated solution has been developed in order to provide nursing personnel scheduling over the Internet as a service [43]. This application involves a graphical user interface, implemented as an ActiveX component, and a wrapper program that executes the nursing personnel scheduling algorithm while it translates the input and output files into XML-formatted documents. The user interface, apart from gathering the input information and displaying the results, also permits certain rescheduling steps. The Execute SPC has been configured to utilize the wrapper on the appropriate computing platform, as well as to transfer several useful files for the execution of the algorithm.

**KTEL.Sched**

The KTEL.Sched application was also created as a diploma thesis project and its purpose is to provide a solution to the difficult combinatorial problem that has to be solved every day in order to produce the daily bus and driver scheduling for a typical bus company in Greece. A quick heuristic scheduling procedure named Quick Shifts (QS) has been developed for the solution of the
combined bus–driver problem, along with a column generation procedure called Column Generation and Quick Shifts combination (CGQS) [7]. Based on these algorithms, a Windows console application was also developed with plain-text input and output file interface. This algorithm was used as the core algorithmic part for the creation of an Internet-based service for the creation of daily bus and driver schedules [44]. This solution involved an efficient user interface implemented as an ActiveX component and a wrapper executable for the controlled execution of the scheduling algorithm with translation of input and output files into XML format. By the use of the wrapper the scheduling algorithm in effect uses XML files for input and output purposes, since SchedSP provides significant advantages for such files. The role of the graphical user interface is to gather all input from the user, create the input files for the scheduling algorithm, instruct SchedSP to execute the scheduling algorithm through the wrapper and then present the results back to the users. The Execute SPC has been configured to utilize the wrapper on the appropriate computing platform, as well as transfer several useful files for the execution of the algorithm.

High-school scheduling

The third scheduling solution based on SchedSP provides the creation of teacher schedules for a Greek high school as an Internet service. In typical Greek high schools, the creation of timetables is a very time-consuming task, which is usually done manually. The discrete and combinatorial nature of the high-school scheduling problem makes its solution quite difficult and in fact the problem is NP complete. An integer programming approach has been adopted for the solution of the problem, leading to high-quality solutions which satisfy the problem definition and constraints [4]. The scheduling prototype of this approach is running on an HP-UX machine and also utilizes the ILOG CPLEX library [46]. The input of this type of problem is complex and as a result it is divided into numerous files because of the various relations between the entities described. The output is a single file with the generated schedule. For the provision of this algorithm as an Internet service, a solution is currently being developed. The main component of this solution is a graphical user interface, implemented as a Java applet. The main difference is the use of the XMLTrans SPC instead of the wrapper used in the previous case studies. The Java applet gathers the user input, creates and updates the corresponding XML files in the SchedSP file space, and requests the execution of the scheduling algorithm. Prior to execution, the applet requests the transformation of the XML input into the plain text files needed by the algorithm. The Execute SPC is configured to transfer these files at the execution site, and utilize only machines based on the HP UX operating system. At the end of the solution process the applet having knowledge of the custom format of the output presents it to the user in a graphical manner. The applet also allows for certain re-scheduling steps.

CONCLUSIONS

In this paper, the possibility of utilizing existing scheduling software as application services available over the Internet is presented. The goal is to allow distant and possibly computationally disadvantaged users to utilize both the software and the hardware capabilities of an ASP. Ease of use and support for these scheduling applications is one of our primary goals. In order to achieve this, a middleware framework based on SPCs is proposed and implemented. The framework architecture is designed to allow for growth both in terms of users using a particular application as well as for the addition of new
scheduling application services. The SPCs and their capabilities were designed so that the user can manage the whole scheduling process lifecycle in a manner similar to a traditional local scheduling system. Our goal is to maintain the essence of such applications although the distant user is required to maintain only a personal computer with minimal Web browser capabilities. Thus, SchedSP provides a reliable middleware framework for the creation of Internet-based scheduling applications for various interface developers, which would not have been possible otherwise.

SchedSP can also be used for non-scheduling applications if the interaction between the user interface and the core of a particular application is not continuous and there are significant computational steps that do take place. More experiments in this direction will be performed in the future.

A central weakness of the service-providing concept that also needs to be further examined and elaborated involves the security risks of the ASP process. Many researchers all over the world are presently tackling these issues and it appears that we will all have a safer Internet interaction available in the near future. The use of the XML Web Services protocols, SOAP, WSDL and UDDI, will also be investigated, although it appears that for this class of applications, which require special purpose local clients, the XML protocol suffices. However, if SchedSP were to fully adopt the XML Web Services specifications, the development of new application-specific SchedSP clients would be further simplified and generalized.

ACKNOWLEDGEMENTS

We want to thank the people who created SchedSP-based client applications for their recommendations on the various SchedSP features, their bug reports and their patience in working with SchedSP while it was under development. More specifically, we would like to thank Emmanuel Ntanos and Michael Mavrommatis, both undergraduate students at the Department of Electrical and Computer Engineering of the University of Patras. We would also like to thank Vassilios Barkayannis, PhD candidate in the Computer Systems Laboratory, for recommendations and work on the client-side of SchedSP. Finally we would like to thank the anonymous reviewers for their helpful comments and creative suggestions.

REFERENCES