A Generic Legality Checker and Attribute Evaluator for a Distributed Enterprise Environment

Christos Goumopoulos, Panayiotis Alefragis, Kleanthis Thrampoulidis, Efthymios Housos
Department of Electrical & Computer Engineering, Computer Laboratory
University of Patras, GR-265 00 Rio Patras, Greece
{goumop,alefrag,thrambo,housos}@ee.upatras.gr

Abstract

The present state of communication networks with respect to speed and reliability and the recent growth of distributed applications have created a need for a global enterprise solution to the legality checking and attribute evaluation requirement. Traditionally, the mainframe systems provided the cohesion of all the processes with respect to the company regulations. When decentralized systems and applications became widely used the legality checking mechanism lost its central role and became a necessary component for every decentralized system. In this paper a methodology to reconnect these systems with respect to their legality checking and attribute evaluation needs is presented. A generic Legality Checking system has been developed and integrated with scheduling systems of the airline domain. It is shown that the client-server model adopted can bring back in a flexible manner the lost homogeneity of the central legacy systems.

1. Introduction

Manpower scheduling and administration is a difficult and time consuming process [10]. The situation is further complicated from the fact that the schedules must satisfy many and sometimes intricate operational constraints. All airlines, for example, must conform to a complex set of union, company and governmental rules and regulations [9]. These rules vary by crew type (pilot or flight attendant), crew size, aircraft type, and type of operation (domestic or international). Work rules, as an example, concern duty periods and rests. A stringent union rule specifies maximum duty period length, which varies between 14 and 16 hours. Other duty rules govern the maximum flying time allowed and the maximum number of flights permitted. The governmental regulations minimize crew fatigue and ensure passenger safety. Minimum rest requirements are tied to the flying time scheduled in a moving 24-hour window. In addition, the after the actual event costing and reporting involved, requires the use of complex rules and formulas and must be also supported by specific rule knowledge. This is due to the fact that changes of the planned work might have happened during the actual operation of the schedule.

Scheduling computer applications are of primary importance because the cost of the human resources is extremely high. For instance, in the airline industry crew costs are the second largest operating expenses after the fuel costs. In recent years, many of the European airline companies have invested in automatic tools for resource planning and scheduling [14]. However, resource management is quite complex and none of the computer systems which currently exist has been designed to address the total problem. It is therefore necessary to divide the resource management task into more manageable components and use special applications and different computers for the various components. Working with Lufthansa German airlines, in the context of the DAYSY Esprit project [5] it was discovered that at least six different systems are employed by several departments for planning purposes [6], such as crew planning and scheduling, aircraft scheduling and real time rescheduling that do require the evaluation and testing of the rules (Figure 1). Since these systems were developed by different vendors and by the airlines themselves they do not utilize the same legality checking approach. There is no central system to provide a common mechanism for the legality checking needed for all the applications. As a consequence many rules have to be replicated and implemented differently and the management of the rules becomes painful.

In this paper a *global enterprise legality server* is proposed, that scheduling applications and other support systems (i.e., accounting, reporting, spreadsheets, management tools) which require the rules, could utilize independently of the hardware platform, database and operating system used. The system follows the client/server model. Clients may connect concurrently

0-8186-8538-7/98 $10.00

with the server and use its legality checking and attribute evaluation services. Each client may load its own selected rule-set and utilize its own database system for retrieving application and problem specific information. The experience and lessons of the airline industry segment should be directly applicable to other industries with complex rules for their man-power organization and planning (e.g., manufacturing, distribution, transportation, construction and various public agencies).

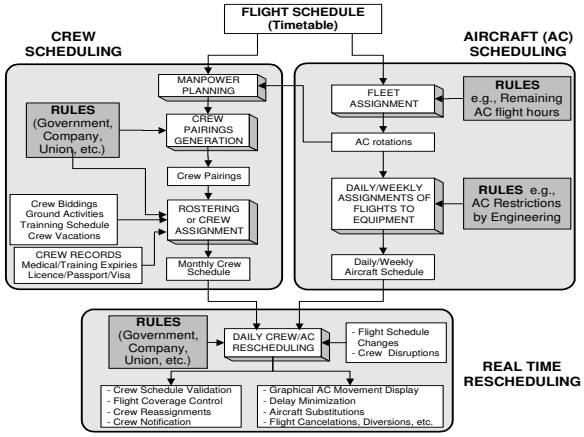

Figure 1. Resource planning and rule checking requirements in airline industry

The rest of the paper is organized as follows. Section 2 introduces the generic Legality Checking and Attribute Evaluation system. This system is currently being used in production by Lufthansa for the legality checking component of the rescheduling process. In Section 3, the abstract database access feature and the capability of a user programmable activity model are presented. The integration of the system with scheduling and other personnel management applications is explained in Section 4. Metrics of the system are given in Section 5. Finally, conclusions with emphasis on the progress and future directions for the improvement of such an environment at Lufthansa are discussed, in Section 6.

2. Legality Checking and Attribute Evaluation (LC) System

Most of the existing computer scheduling systems check the legality of the produced solution using a few external parameters for the customization of the system and include the implementation of the rules within the application software. However, since labor rules are continuously changing, there is a need for a high level domain specific language in order to express and manage these rules. Two systems that use a special purpose language for the expression and subsequent management of rules are presented in [3] and [16]. In this case, using an application specific language as an interface, 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 what-if scenarios and test for rule extensions and additions without changing the application programs.

The development of a generic LC system entailed the identification of the information that would enable the domain-specific planner to easily express the regulations of the problem. The most difficult part of the development of the LC was the acquisition of this knowledge. Applying domain analysis to the scheduling problem domain an *object meta-model* was created [16], the main part of which is depicted in Figure 2. 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 provides the infrastructure for their individual calculation which is responsible for the attribute evaluation flavor of the produced system.

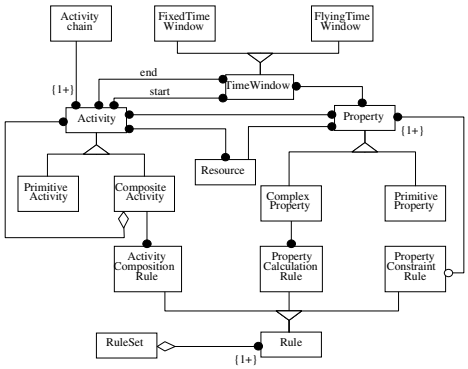

Figure 2. Object meta-model of the LC system

For the defined meta-model to be easily applied to a wide range of application domains a two step process must be followed (Figure 3). In the first step, specific problem domain experts (airline experts in our case) apply domain analysis to create the airline object-model as an instance of the generic meta-model. This model contains declarations of airline generic activities, proper-ties, rules and problem domain keywords. In the second step, the specific application rule manager (e.g., Lufthansa's rule manager) specializes and refines the airline object-model in order to produce Lufthansa's application object model. However, from the users point of view, the definition of these object-models in terms of a general purpose

programming language is very difficult and often requires external support. This was the motivation for the definition of the high level rule language REDOM (ex DAYSY Rule Language) [15,16].

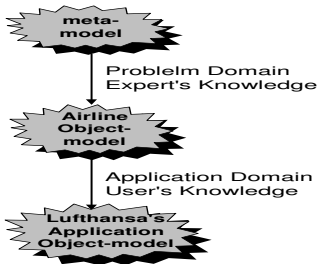

Figure 3. Generation process of the application's object model

Using REDOM the rule manager easily transforms the generic model to an application specific model. In the airline domain, for instance, some specific entities to be modeled are flight activity, shift activity, maximum duty time regulation, minimum rest time per 24 hours regulation, etc. REDOM language does provide all the appropriate lexical and syntactical structures, as well as the appropriate semantics [7] for the instantiation of the meta-model of Figure 2.

For the translation of the REDOM programs to executable code a compiler was created. The REDOM compiler implements the front end of the REDOM language translation and produces an intermediate C++ code from the original REDOM source. C++ provides high-level abstractions with the efficiency of a low-level language like C. The back end of the compilation process is assigned to the corresponding C++ compiler of the target machine. This scheme enables portability of the REDOM compiler, and the use of optimization techniques provided by C++ compilers. The produced object code is finally linked with the *LC Interface library* and the *LC Kernel library*, generating the LC run-time system (Figure 4). The LC Interface library implements the message protocol for the communication with the scheduling applications while the LC Kernel library contains the fixed part of the LC.

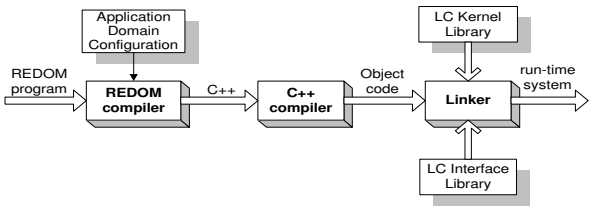

Figure 4. Building the LC run-time system

The basic components of the LC architecture are the *Message Dispatcher*, the *Activity Recognizer* the *Attribute Evaluator*, the *Rule Checking Mechanism* and the *On-line Rule Data-Part Manager* as shown in Figure 5.

The Message Dispatcher component is the interface with the external world. It receives requests from the scheduling application and sends back responses through a message protocol. The requests are satisfied by calling the corresponding methods. These methods constitute the LC API, that provides the legality checking, the attribute evaluation and the on line data-part management services.

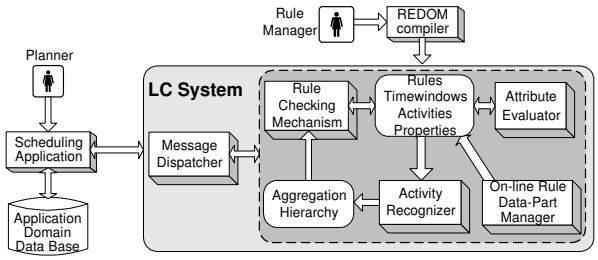

Figure 5. LC system architecture

With the receipt of the schedule, the rule system creates all the activity objects of the aggregation hierarchy corresponding to the activity composition rules. Activity composition rules of the form "create a shift object when there exist 11 hours of rest period between two flight objects" determine the shape of the aggregation hierarchy and are part of the *application domain configuration*. The properties and the constraints, associated with each activity object, have already been defined by the rule manager in terms of the REDOM language. After the aggregation hierarchy is completed, the rule system performs the attribute evaluations and the constraint checking. The On-line Rule Data-Part Manager supports the on-line manipulation of the rule parameters, enabling planners to test alternative what-if scenarios without recompiling the rule set.

3. Abstract Data Access and Application Domain Configuration

The Rule Checking Mechanism of the LC system needs information concerning domain activities and resources (e.g., the arrival time of a flight, the qualifications of a crew member, the type of an aircraft), in order to calculate properties and check the constraints. This information is stored in application specific data bases. A global enterprise LC system should be able to work with different data base environments because different computer systems in the same enterprise may access different data bases. The transition to different application

domains would be also much more painful if the system was tightly coupled with a particular database management scheme. This was the motivation for the abstract data access mechanism of the LC.

A number of identifiers are designated by the rule manager as *keywords* of the specific problem domain. Keywords are used to easily access information for domain specific activities and resources, which are located in a particular database. They are declared as part of the corresponding activities, during the creation phase of the specific problem domain object-model and are defined during the creation phase of the application object-model. They are supplied at run time by the scheduling process through a message protocol, providing the LC with the necessary independence from the database scheme of the client system.

The values of the keywords may be retrieved from any possible database management system as long as an API is supplied from the user of the particular data base. This kind of an API should consist from a set of functions that given a keyword name and an activity identifier, return the value that is stored in the database. A prototype declaration example of the main function for the retrieval of keywords might be:

```
value_type   get_keyword(String <keyword_name>,
                          TypeAct <activity_type>,
                          int <activity_identifier>);
```

Figure 6, shows the retrieving mechanism of the keyword values. Using the concept of keywords the same low level representation of a common rule-set handled by the LC server, can be used by different processes that use different database systems.

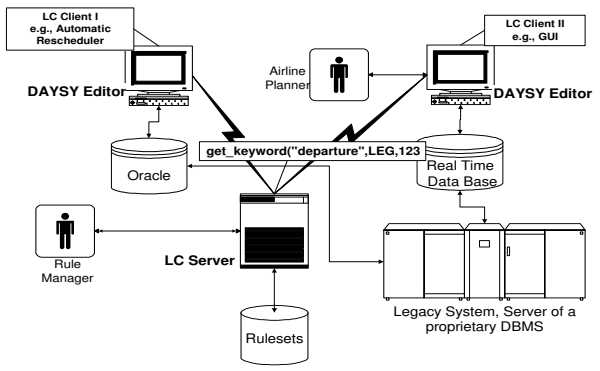

Figure 6. Retrieving mechanism of keywords

For a client/server architecture where the LC system is the server process and the client process creates sets of activities to be evaluated and/or tested, the keyword values can be supplied by the client process. The LC server requests keyword values from the client process. The client process must use a keyword server that acts as an intermediate layer between the LC system and the database which creates an abstract retrieving mechanism of the keyword values. When working in a network environment and the global access of information occurs frequently the latency of the network is a major computational constraint. A *caching mechanism* has been developed in order to reduce the delays created by the distant transfer of data. The LC server stores the data retrieved by a request over the network and if a new request for the same data appears, the cached data is returned.

The application domain configuration is realized mainly through a configuration file that supplies the typical structure of each primitive and composite activity. This file defines the object model of the user's problem domain. The *ACTIVITY* reserved word declares an activity type that is associated directly or indirectly with rules. An activity declaration consists of component activities, neighbor components, keyword names, complex property names, and names of applied constraints. For the airline domain typical activities declared in the configuration file are: leg, shift, rotation, roster, simulation, vacation, training, standby, rest etc. For example, shift is an activity that the user can extend with new properties and constraints using the inheritance capability of the REDOM language. Thus, the user has the ability to define new activities and incorporate new keyword names in the REDOM language. A typical activity description of the configuration file follows.

```
ACTIVITY shift
        NEIGHBORS:
                shift, rest;
        COMPONENTS:
                leg, simulation, training;
        KEYWORDS:
                ac_type: string, departure: tabs;
        PROPERTIES:
                duty_start   : tabs,
                duty_end     : tabs,
                duty_period : trel;
        CONSTRAINTS:
                max_duty_time;
END
where tabs, trel, string are built-in REDOM language data types.
```

4. Enterprise-wide LC Engine Sharing

The LC system presented in section 2 has been integrated at Lufthansa with a Graphical User Interface (GUI) for manual planning [4] and with an Automatic Rescheduling System (ARS) [1] for automatic planning. The client/server model for network applications has been used. The interaction between LC clients and the LC server is based on a three-layer protocol stack (Figure 7).

The LC application layer protocol reflects the LC API [8], that provides for the attribute/property evaluations, the legality checking and the on line rule data-part management services. Examples of such services include:

- Open a line of work (low).
- Close a low.
- Add new activities or remove activities to/from the low.
- Check the legality of the low.
- Get the value of a property (e.g., trip cost, pay cost, etc.)
- Turn on/off a rule.
- Update a rule parameter value (e.g., rule limits, etc.)
- Get the aggregation hierarchy created so far.

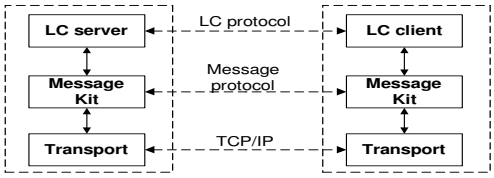

Figure 7. A 3-layer interaction model

The underlying message protocol determines the structure of the exchanged messages to transfer commands and data. Through these messages the client can send the activity data, ask for legality checking, etc. After sending a request the client will get a response, e.g., the result of the legality check, the value of an attribute evaluation, etc. The server, after receiving a message request from a client and in accordance with the message command, it calls the C++ methods of the LC server. The Message Kit layer provides an API to the application layer with services such as create an endpoint, send/receive a message command, send/receive a message response, connect/disconnect to LC server, reset communication, time-out communication.

Finally, the message protocol is based on the conventional TCP/IP protocol. The transport socket interface [13] is used as the inter-process communication mechanism. This low level mechanism was preferred because the primal concern was for the performance and because of its availability in every hardware platform used.

Figure 8 depicts the distributed legality checking model. The LC server presented up to now is represented with the darker box. The complete integration environment requires some additional components. First of all there is a daemon process, named as *LC Daemon*. This is a concurrent server, it listens to a well-known endpoint and waits for connections from client processes. When there is a connection request the concurrent server invokes another process to handle the client request.

A client request is served initially by a process, named as *LC Agent.* Its purpose is to set-up the legality session and start the execution of the actual LC server instance. This is necessary since a client may select among several rule-sets for the legality checking process. The execution of the LC server instance may be either local or remote depending on the computational and response needs of the calling client. For the interaction with the client a special protocol, named as *agent-protocol,* has been defined. The LC agent has basically the following responsibilities:

1.To authorize the client and control the right access of the services. For example a user may be able to check the legality of the schedule but forbidden to alter the data parameters or access attribute evaluation data for security reasons.

2.To get the activity configuration file specifying the problem domain.

3.To get the REDOM rule-file description submitted by the client.

4.To invoke the rule translator in order to create, the runtime instance of the LC.

5.The management of the LC server instances, i.e., loading, unloading, deleting, dynamic endpoint assignment of the various specific LC servers. The client may request the loading of an already compiled rule-set by sending its name. A dynamic endpoint assignment is utilized for the loading service.

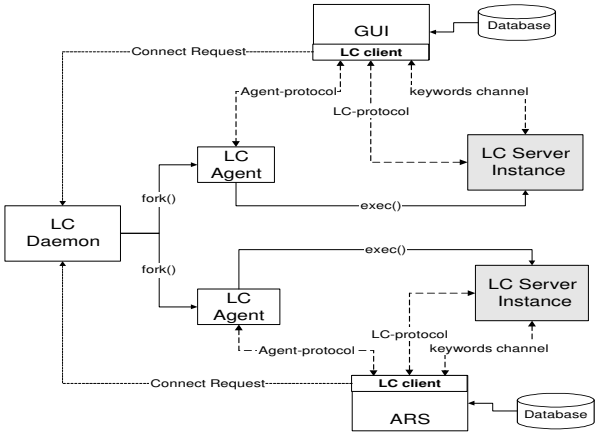

Figure 8. Distributed legality checking model

The actual legality checking and attribute evaluation services are provided by the *LC server instance*. The approach of separating the management services from the legality services improves the efficiency and the extensibility of the system as the management of operations, not involving the legality checking operation itself, are kept out of the LC server. This makes the LC server responsible for the management of multiple activity chains of the same user and their legality checking.

For the evaluation of the activities properties, keyword values are sometimes necessary. Each time a keyword value is needed the LC server instance requests it from the keyword server located at the client side. For each request the keyword name and the activity identifier are passed to

the keyword server. The keyword server interacts with the local DBMS and returns the corresponding value. For the implementation of this scheme, a secondary endpoint is required for the communication of keyword values. The LC server instance has to establish a secondary channel with the client so that it can access the keyword values.

The distributed LC model allows the existence of redundant LC server instances. Redundant servers (LC server instances of the same rule-set) are desirable to allow for load balancing and failure resilience. Typically, each LC client is serviced by a distinct LC server instance. Currently, there is one ARS, and up to four GUI processes that have to connect to the LC server. In particular, at the set-up phase of the real time rescheduling system, more than 10,000 lines of work must be checked for legality. This requires more than half of a computing hour if a single LC server (and client) is involved. The existence of multiple LC servers significantly reduces the computation time. In case of a failure the mechanism that requests the service will detect the LC server failure and transfer through the LC agent the request to another server if possible. If no alternative server exists, the client is informed that the service is unavailable and a new LC server must be started.

In Figure 9, a possible interaction scenario between the Legality Checking System and an LC client, is presented. In step 1, the client submits a REDOM rule-file and the LC agent invokes the rule translator to create the low level rule binary. This low level rule binary is then linked with other library modules in order to create the specific LC server instance. The result of the compilation phase is transmitted to the client, with the report of possible errors. If the compilation phase is successful the client may send a request (Step 5) for the loading of the new LC server (Step 6). The client then connects with the LC server (Step 8) through the assigned port number returned at Step 7. Next, the client sends the activity chain to be checked (Step 9). The client can then send a request in order to start the legality checking process (Step 11) of the activity chain and the legality checking phase is entered (Steps 12, 13, 14). Afterwards, the legality checking result is reported to the client (Step 15). The client may then send some other activities to be checked (Step 9) or unload the LC server (Step 17).

The employment of a rule checking and attribute evaluation system as an enterprise-wide legality/evaluator server for all personnel related computer systems of an airline company is feasible and practical. Replacing the built-in legality checking procedure of existing scheduling applications with an association to the enterprise legality server has primary advantages. First of all the dynamic modification of the rules without disturbing and risking the integrity of the application. In addition, this methodology provides a single system for maintenance and support and a common language to express all the rules. The REDOM language has been proven in practice capable of expressing all the necessary rules in the Lufthansa operating environment.

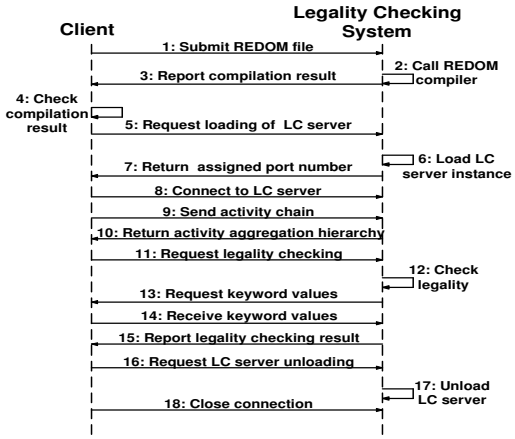

Figure 9. A possible interaction scenario

5. Metrics

The distributed computing environment consisted of a network of HP9000-715 workstations interconnected both with a 10 Mpbs Ethernet and a 100 Mbps FDDI network. The implementation used the TCP socket interface as the inter-process communication mechanism. In addition, the system makes use of Unix-domain stream sockets [13] as an alternative to TCP for local communication between the client and the server, to improve latency, typically by a factor of up to five. For the benchmark process, an implementation that uses the LC system as a library linked to the client application, was also available. The LC client application used, was a graphical user interface. The user creates chains of activities and then sends them to the LC server for legality checking.

In order to check a line of work (low), the client application and the LC server have to exchange a minimum of 8 messages. These messages specify the requests and the corresponding responses for operations such as to open a low, to add the activities, to check the low and to close the low. Additional messages may be exchanged for accessing the keyword values or in the case of incremental legality checking, i.e., when the activities of the low are added one by one, and several intermediate legality checks must be performed. The size of the messages to be exchanged is always small, less than 1024 bytes, which makes the latency of the interconnection network the dominating factor of the communication overhead.

Table 1 gives the performance results of the distributed LC system. We report the time to check four typical line of works of different sizes. The complexity of the rule-set in use and the number of the activities contained in the

low determine the computation time of the legality checking operation. The measurements have been done after the TCP connection has been established. In addition, in every case, we have made all the necessary keyword values available locally in the cache, before checking the legality.

*times in ms	TCP Ether	TCP FDDI	Unix Streams	Lib	TCP overhead Ether	TCP overhead FDDI	Streams overhead
low1 (23)	519*	515.4	509.9	506	2.6%	1.8%	0.7%
low2 (15)	366.9	363.4	357.6	354	3.6%	2.6%	1%
low3 (13)	280.8	277.3	271.4	268	4.7%	3.5%	1.3%
low4 (11)	212.7	209.3	203.4	200	6.3%	4.6%	1.7%

Table 1. Metrics of the distributed LC system

For applications like the GUIs or management tools where manual operations are performed the TCP overhead can be acceptable. However, for computationally intensive real-time systems like the automatic re-scheduler or the automatic crew schedule planning system, a local inter-process communication mechanism is necessary to reduce the overhead. Low latency networks [2] and optimized message passing implementations [12] which avoid operating system intervention or complicated protocol layers of traditional local area networks can make the network-wide server more viable even for the computationally intensive applications.

6. Conclusions

The use of decentralized workstation based applications has been an effective alternative to the mainframe model for most industrial environments. However, this decentralized model, in contrast with the mainframe era, suffers when global concepts and rules must be re-implemented in several applications and computers. In the airline industry, in particular, there exist various workstation based applications (e.g., crew planning, aircraft scheduling, real time rescheduling) that need to evaluate a set of company and state regulations in various instances of their solution process. In this paper a methodology to unify the attribute evaluation needs of several applications and to provide a global legality checking service was presented. The main advantage of the proposed client-server approach is the increased reliability of the legality system and the unification of the rule implementation and storage characteristic. Some applications could have a local instance of the legality server if this is required by its intense rule computational needs in order to avoid the potential communication overhead.

This approach to attribute evaluation and legality checking is currently being tested for use at the Lufthansa crew scheduling department. The object oriented design and implementation of the server makes natural its adoption by a CORBA (common object request broker architecture) [11] based distributed computing environment. A distributed object computing framework will enable the interworking between workstation-based applications at higher levels of abstraction and components to collaborate more efficiently and transparently. This will leverage the usability of the system as other user programs may use the CORBA services to have a legality checker module as if it was a local object. The current plans also involve the creation of a JAVA based interface so that the system can be used for the new Internet based work assignment selection by the pilots of an airline company.

References

[1] G. Baues et. al., *DAYSY Automatic Rescheduler: Detailed Design Specification*, DAYSY EP8402 TR D.5.3.4, Cosytec SA, Orsay Cedex, France, 1996.

[2] N.J. Boden et. al., "Myrinet: A gigabit per second Local Area Network", *IEEE-Micro*, 15(1):29-36, Feb. 1995.

[3] CARMEN PAC 5.0 - User's Reference Manual, Carmen Systems AB, Gothenburg, Sweden, 1997.

[4] L. Chudant et. al., *DAYSY Rotation Editor: Man-Machine Interface Specification*, DAYSY EP8402 TR D.7.4.2, Sema Group SA, Paris, France, 1996.

[5] DAYSY consortium, *Technical annex for the Esprit project 8402: Day-to-day resource management systems,* DAYSY, January 1994.

[6] Deutsche Lufthansa AG NE 4, *Crew Management Metrics*, DAYSY EP8402 TR D.8.2.1, Frankfurt, Germany, 1996.

[7] C. Goumopoulos et. al., *Syntactic and Semantic Definition of DAYSY rule language*, DAYSY EP8402 TR D.4.2, Patras, Greece, 1996.

[8] C. Goumopoulos, and P. Alefragis, *Legality Checker C++ Application Programming Interface version 1.1*, DAYSY EP8402 TR D.4.4.2, Patras, Greece, 1997.

[9] G.W. Graves et. al., "Flight Crew Scheduling", *Management Science*, vol. 39, no. 6, pp. 736-745, June 1993.

[10] Nanda, R., and J. Browne, *Introduction to Employee Scheduling*, John Wiley & Sons, New York, June 1992.

[11] OMG, *The Common Object Request Broker: Architecture and Specification*, revision 2.0, 1995.

[12] S. Pakin et. al., "Fast Messages: Efficient, Portable Communication for Workstation Clusters and MPPs", *IEEE Concurrency*, vol. 5, no. 2, April - June 1997.

[13] Stevens, W.R., *Unix Network Programming* Prentice-Hall, 1990.

[14] Suhl, L., *Computer-aided scheduling - an airline perspective*, Gabler Edition Wissenschaft, Wiesbaden, 1995.

[15] K. Thrampoulidis, C. Goumopoulos, and E. Housos, "Rule Handling in the day-to-day Resource Management problem: an Object-Oriented approach.", *Information and Software Technology*, vol 39, pp. 185-193, 1997.

[16] K. Thrampoulidis et. al., "REDOM: An OO Language to Define and On Line Manipulate Regulations in the

Resource (Re)Scheduling Problem", *Software Practice and Experience*, vol 27, no 10, pp. 1135-1161, 1997.