Regulation Management in Resource Scheduling Problems with fREeDOM System

Christos Goumopoulos
Dr. Electrical and Computer Engineering

2nd AMORE Seminar, 30 Oct - 3 Nov, 2001,
Patras, Hellas

Outline

- A few words about LYSEIS Ltd
- Resource Scheduling in Airline Industry
- Modeling Regulations with fREeDOM
- Complete Regulation Management
- Conclusions

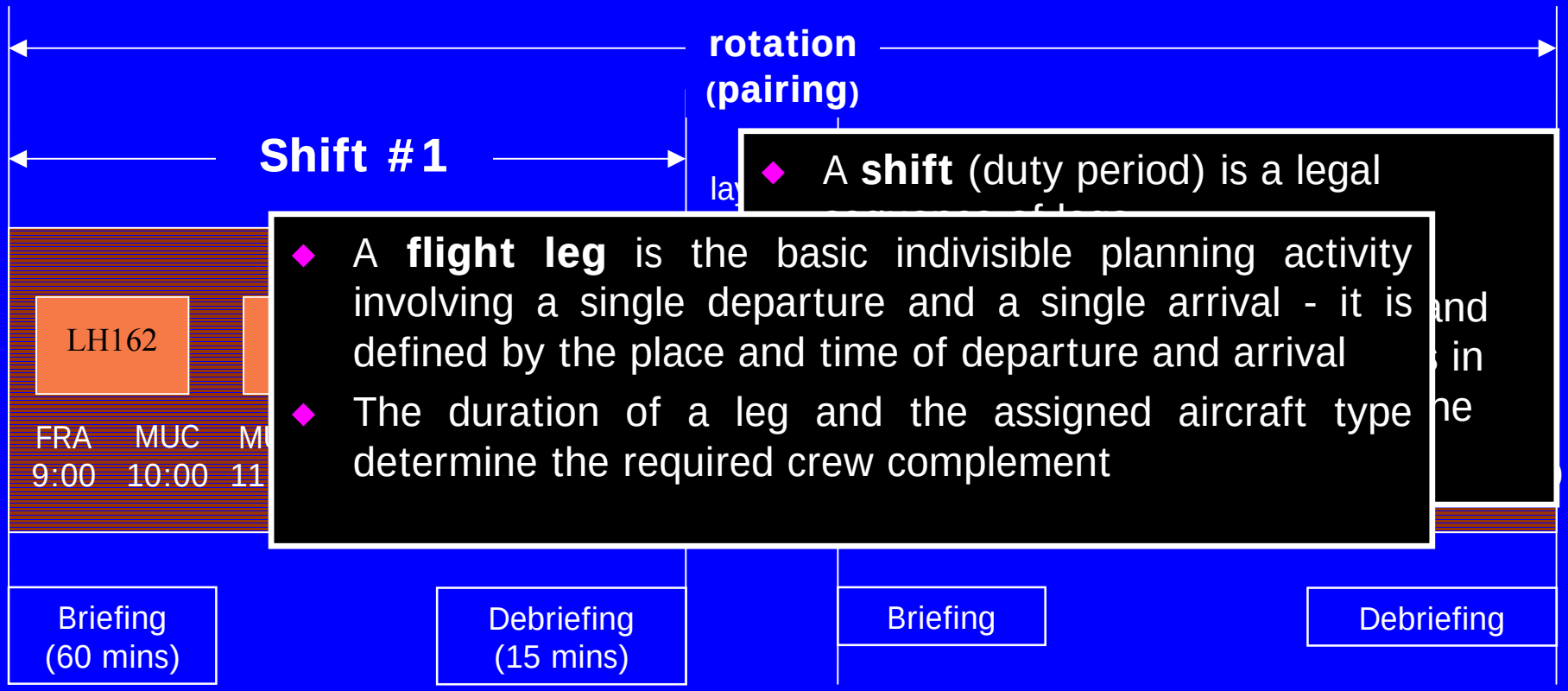
LYSEIS Ltd AITS

- Main figures
  - Spin-off company established in 1998
  - 3 CS PhD associates + external collaborators
  - Patras Scientific Park building (HQ)
- Products and services
  - fREeDOM system
  - RIDE (Ruleset Integrated Development Environment)
  - Crew Rescheduling System (under development)
  - Shift scheduling for call centers (under development)
  - Consulting for developing resource management systems
  - Technical & educational support

Planning Processes in the Airline Industry

- Timetable construction
  - All the flights the airline decides to operate
- Aircraft scheduling (Fleet Assignment)
  - Optimal assignment of a specific aircraft type to every flight, satisfying various constraints
- Crew scheduling
  - Optimal assignment of crews to every flight, satisfying a large number of regulations
- Day-to-day resource rescheduling
  - Optimal confrontation of unexpected events during the execution of the program, satisfying all the regulations

Terminology



- A **shift** (duty period) is a legal
- A **flight leg** is the basic indivisible planning activity involving a single departure and a single arrival - it is defined by the place and time of departure and arrival
- The duration of a leg and the assigned aircraft type determine the required crew complement

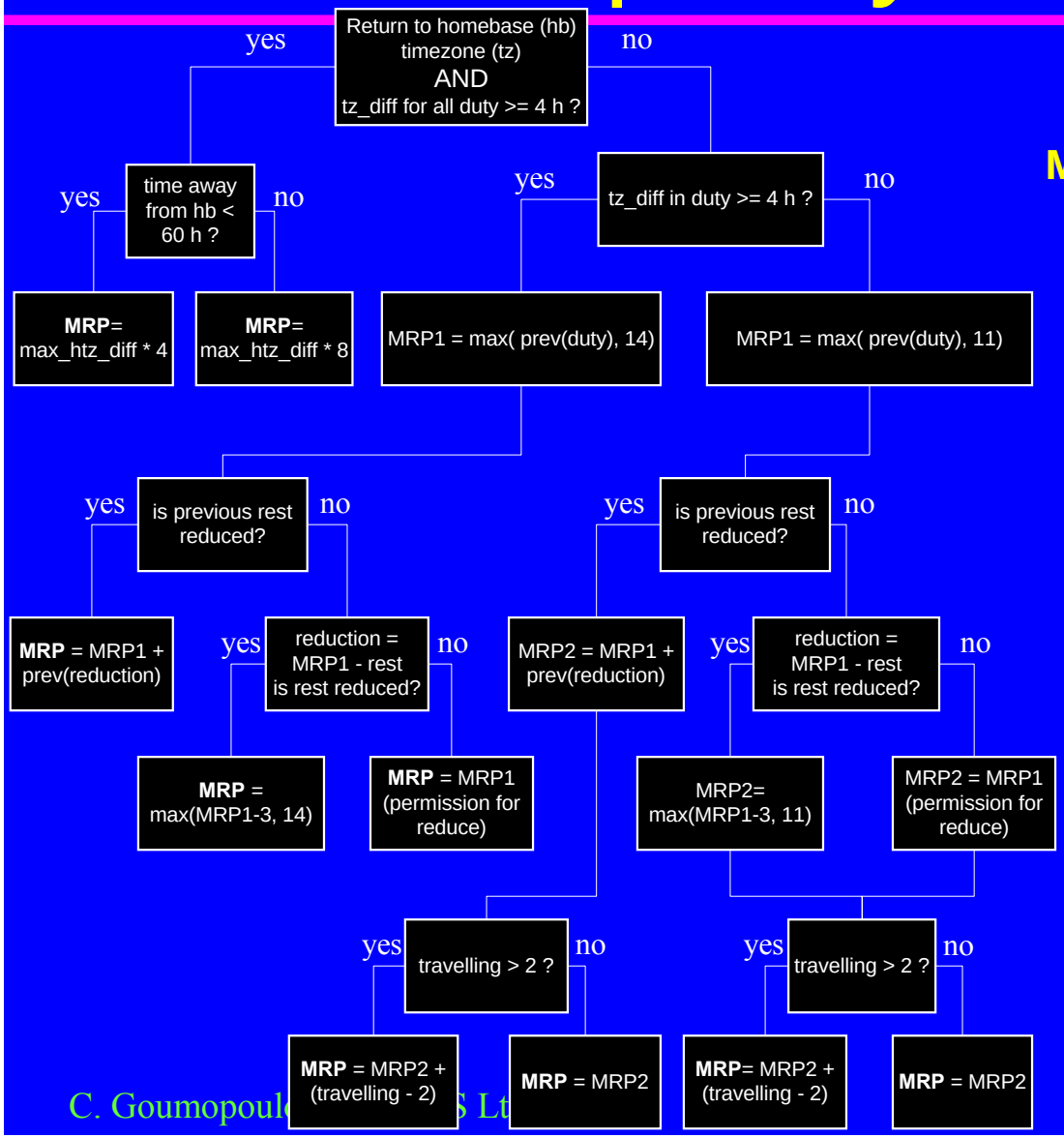
- Every **rotation** departs and returns to the same **crew base** (airport)
- A legal rotation is assigned indivisible to one or more crew members

MODELING REGULATIONS WITH fREeDOM

Regulations to Model

- Fundamental legality rules
  - safety regulations (governmental, international)
  - union contracts
- Operational stability rules
  - company policies
- Quality rules
  - soft rules to improve the quality of schedules
- Rules to improve scheduling application performance
  - special pruning rules
- Complex cost functions for the optimization problems

Rule Complexity Example



Minimum Rest Period Calculation Rule

Depending on factors such as:

- **the duration of the preceding shift**
- **if the preceding shift contains a split (includes a break from 3 to 11 hours)**
- **if the preceding rest period was a reduced one**
- **the traveling time in ground**
- **the timezone difference between the place of starting the shift or the parent rotation and the place ending the current shift**

How to Model the Regulations?

- Hard-coding the regulations in the scheduling application has disadvantages:
  - application integrity risk in case of a rule change
  - changing the rules requires expert programmers
  - large maintenance cost
- Handling the regulations with an autonomous system has advantages:
  - **user-oriented** (less cost, flexibility)
  - **vendor-oriented** (less maintenance effort, application safety, easier deployment to new clients)

The fREeDOM System

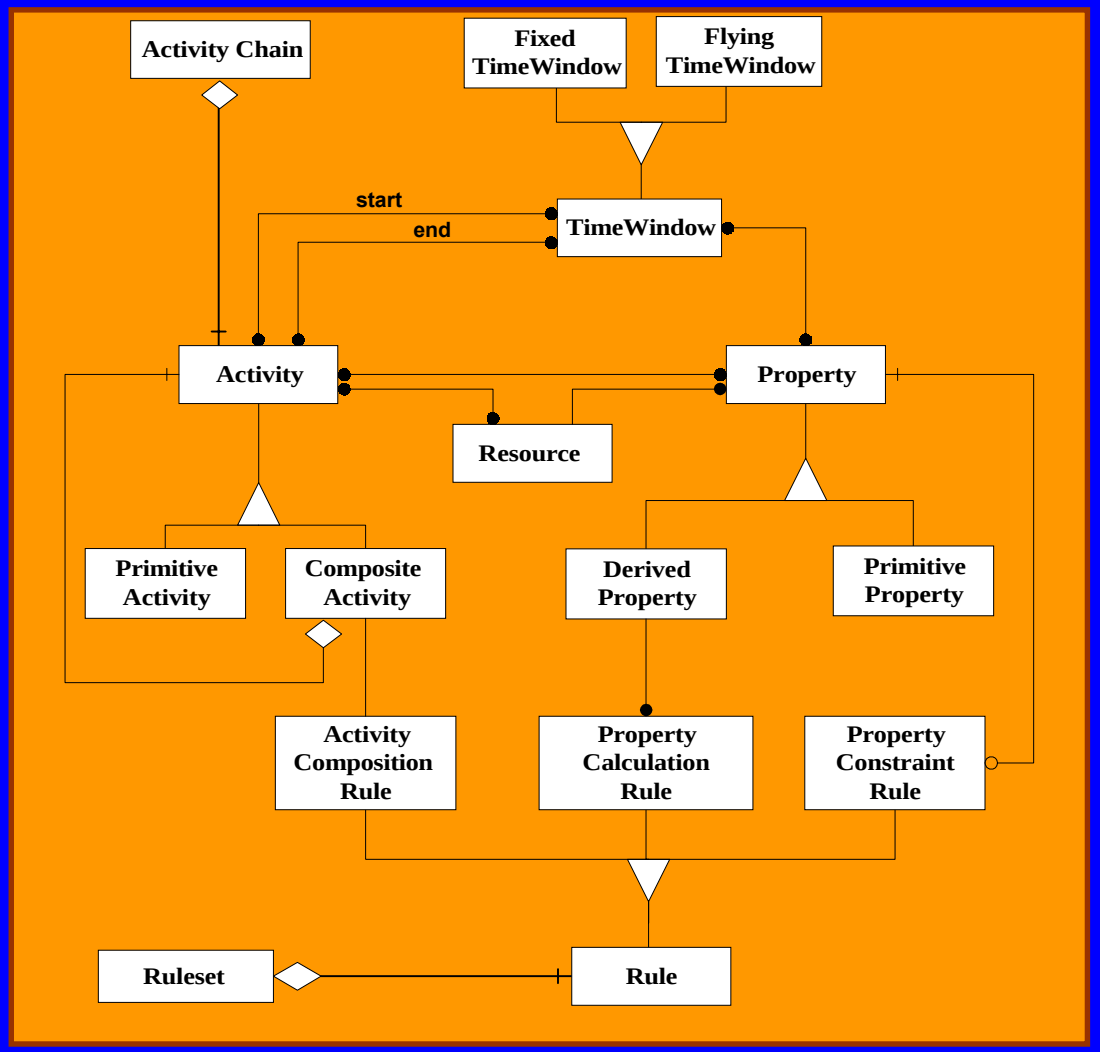
- fREeDOM (**f**ast **RE**gulation **D**efinition and **O**n-line **M**anipulation) is a flexible s/w component for developing Regulation Handling Systems
- Complete regulation management
  - Allows the immediate adaptation of scheduling systems in regulation changes
- Addressed to any company that needs to have its rules under control
  - Expression and update of rules from the end-user
- Prototype system was developed at Computer Laboratory of Electrical and Computer Engineering Department at University of Patras (DAYSY ESPRIT project)

Modeling Language

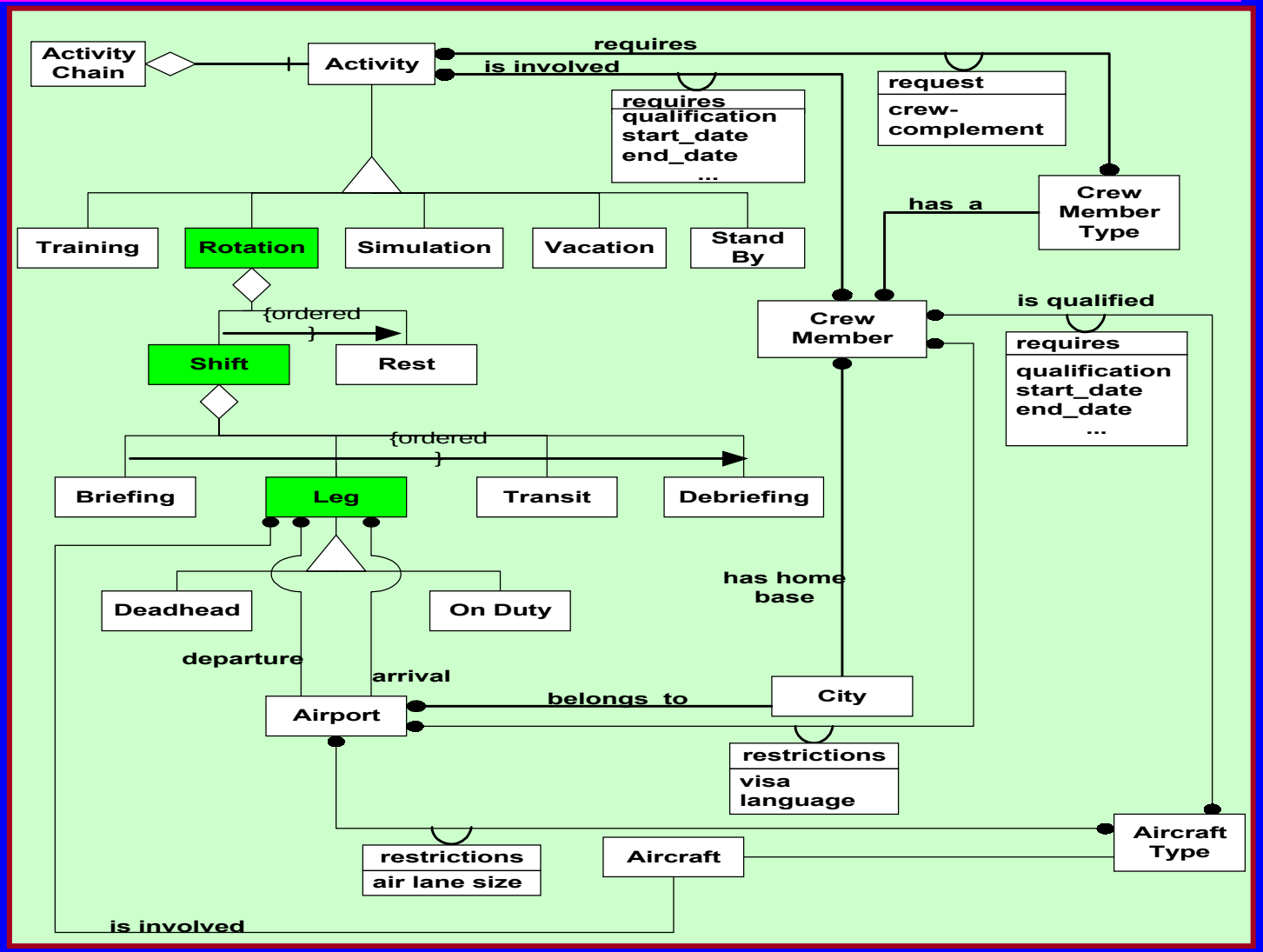
- Declarative special purpose modeling language (emphasizes on WHAT not on HOW)
- High level language semantics closely related to the user terms

fREeDOM Object Meta-model

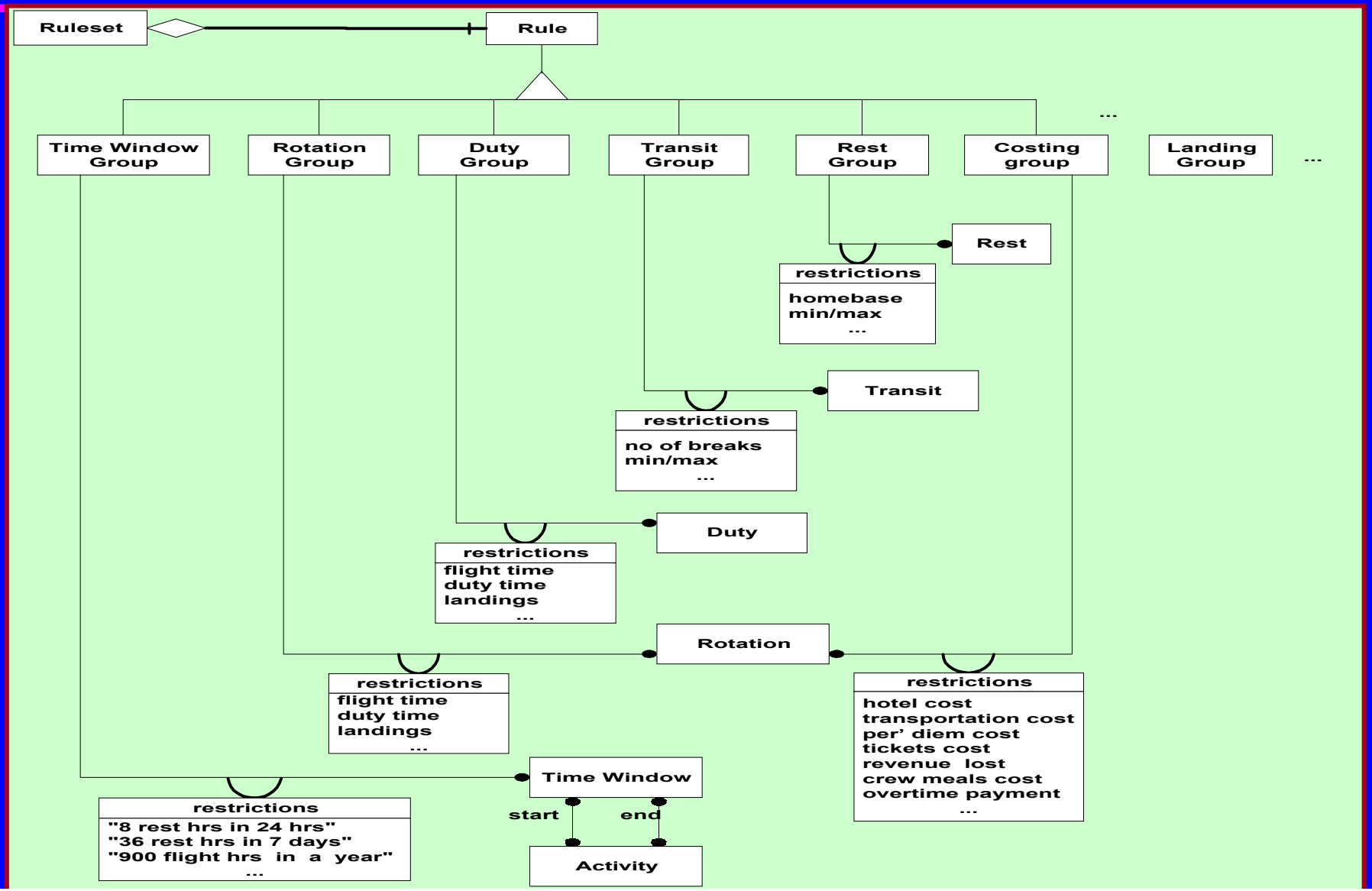
- Includes scheduling problem domain abstractions:
  - Rule
    - Activity Composition
    - Property Calculation
    - Property Constraint
  - Activity
    - Primitive
    - Composite
  - Property
    - Primitive
    - Derived
  - Timewindow



Airline Object Model (part of)



Regulation Categories (part of)

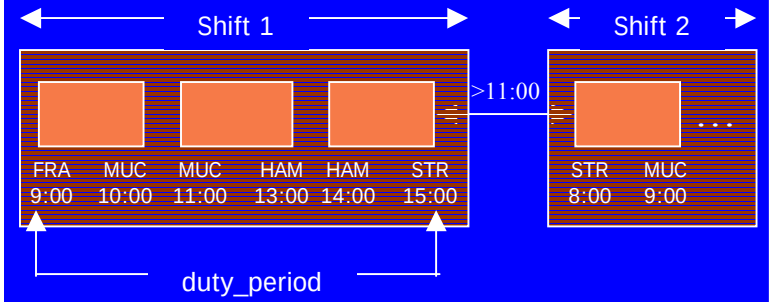


Regulation Modeling Example

Activity Type Declaration

```
ACTIVITY shift
    NEIGHBOURS  : shift;
    COMPONENTS : leg;
    PROPERTIES   : duty_period  : trel;
    CONSTRAINTS: max_duty_period;
END
```

Activity Composition Rule




```
COMPOSITION OF shift
    RULE:  ((departure - (of prev leg arrival) > 11:00)
                                  or
            ((of parent shift (duty_period > 12:00))
                                  and
           (departure  - (of prev leg arrival)  > 08:00)));
END
```

Property Calculation Rule

«shift duration»

```
PROPERTY duty  period OF shift
    RULE:    (of last leg arrival) - (of first leg departure);
END
```

'property calculation expression'

Property Constraint Rule

«Maximum duration of a shift must be 14 hrs»

```
CONSTRAINT max_duty_period OF shift
    RULE:    duty_period <= 14:00
END
```

Data Types

- Conventional
  - integer, boolean, float, string
- Relative time
  - *Example: '2:00'*
- Absolute time
  - *Example: '1 Jan 2001 0:00'*
- Set
  - *Example: SET Hellenic_airports = "ATH", "SKG", "JSI"*

Aggregation Operators

- Calculate a value aggregating a property/expression over all the component activities of a composite activity

  SUM AVG MIN MAX COUNT ALL ANY

  – The range of participating component activities can be controlled with the use of a **WHERE** condition

  – The calculation can prematurely ended with the use of a **WHILE** condition

  – Direct aggregation in any level of the implied aggregation hierarchy

Flying time in a shift →

```
SUM fly_time OVER leg;
```

Flying time in a shift based only on flight legs departing from Hellenic airports →

```
SUM fly_time OVER leg
WHERE (departure_airport IN Hellenic_airports);
```

Flying time in a shift till the first Hellenic airport →

```
SUM fly_time OVER leg
WHILE not (departure_airport IN Hellenic_airports);
```

Specifier Operators

- When the calculation of a value requires referencing:
  - Properties of the first or last component activity
  - Properties of the previous or next activity
  - Properties of the parent activity

  FIRST LAST NEXT PREV PARENT

  - Direct reference to component activities in any level

Specifying the arrival time of the last leg in a shift → arrival of **LAST** leg

Specifying the departure time of the first leg reaching a Hellenic airport →

departure **OF FISRT** leg
WHERE (arrival_airport **IN** Hellenic_airports);

Lookup Tables

- When a property evaluation could make choices among several values that depend on a list of several conditions
- Facilitates handling of data that are subject to change without a need to recompile the source code

Example : *Maximum flight duty time allowed in a shift*

Depends on a) the number of flight legs and b) the starting-time

Defining max. flight duty time in a shift

```
PROPERTY max_fdt OF SHIFT
    RULE: [no_legs] [shift_begin]
    FILE: "Duty_Tables.et"
    TABLE: "Maximum_Duty_Period";
END
```

Number of flight legs

Starting time

#	[1-2)	3	4	5	6	7	>=8	default
07:00-11:59	14:00	13:15	12:30	11:45	11:00	10:15	9:00	9:00
12:00-13:59	13:30	12:45	12:00	11:15	10:30	9:45	9:00	9:00
14:00-15:59	13:00	12:15	11:30	10:45	10:00	9:15	9:00	9:00
16:00-17:59	12:30	11:45	11:00	10:15	9:30	9:00	9:00	9:00
18:00-03:59	12:00	11:15	10:30	9:45	9:00	9:00	9:00	9:00
04:00-04:59	12:30	11:45	11:00	10:15	9:30	9:00	9:00	9:00
05:00-05:59	13:00	12:15	11:30	10:45	10:00	9:15	9:00	9:00
06:00-06:59	13:30	12:45	12:00	11:15	10:30	9:45	9:00	9:00
default	13:30	12:45	12:00	11:15	10:30	9:45	9:00	9:00
#								

TIMEWINDOW Abstract Type

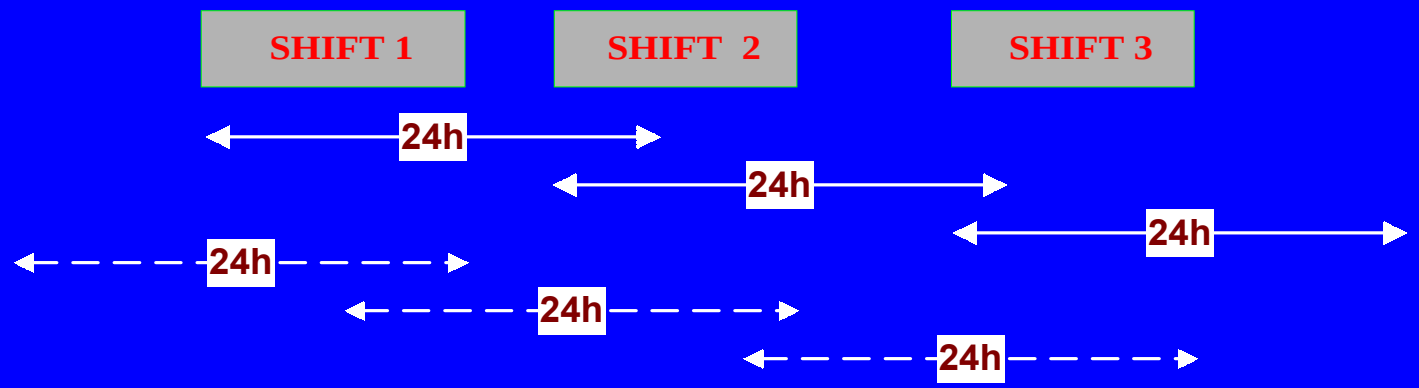
- Abstract concept representing a moving time-period over the activity aggregation hierarchy
  - The underlying mechanism is transparent to the user (declarative programming)
- Imagine it as a virtual activity in the aggregation hierarchy
  - We can associate to them property calculation and constraint rules
- Facilitates the expression of constraints over time periods

  "The crew-member should be granted at least 8 hours of consecutive rest time in any 24 hour period"

  "The flying time of a pilot must not exceed 900 hours per year"
- Two build-in categories
  - *Calendar* Timewindow (calday, calweek, calmonth, calquarter, calyear)
  - *Sliding Activity* Timewindow

TIMEWINDOW Example

- **Regulation Statement** : *In any 24 hour period, the working time of the crew should not exceed 12 hours*.
- There are two types of 24-hour periods to consider (for each type multiple instances are created)
  - (1) The period starting at the beginning of a shift
  - (2) The period ending at the ending of a shift



Source Code in fREeDOM

Declaration Section

```
TIMEWINDOW My24HourWindow
EXPANDS SlidingActivity
   INIT:
            twduration = 24:00;
            twtype = BOTH;
   STEP:
            shift;
   PROPERTIES:
            duty_time  : trel;
   CONSTRAINTS:
            max_duty_time;
END
```

Definition Section

```
RULEDATAPART
    duty_limit_24h = 12:00;
END

PROPERTY duty_time OF My24HourWindow
     RULE:   sum overlap(duty_start, duty_end, TWSTART, TWEND)
             OVER shift;
END

CONSTRAINT max_duty_time OF My24HourWindow
      RULE: duty_time <= duty_limit_24h;
END
```

Defining Quality Rules

Typical constraint rule

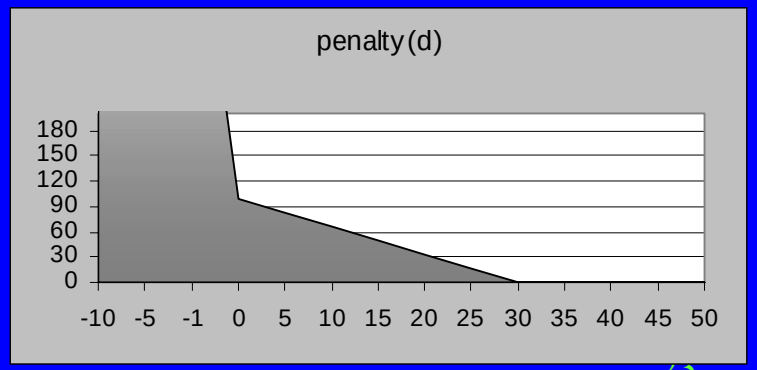
```
CONSTRAINT min_transit_time OF LEG
        RULE: transit_time<=50:00;
END
```

- Quality rules allow the relaxation of constraint limits
- A penalty value is returned when checking a quality rule
- Better quality solutions from scheduling applications

Defining a quality rule

```
CONSTRAINT min_transit_time OF LEG
        GUARANTEE = 0:30;
        BASE = 100;
        RULE: transit_time<=50:00;
END
```

$$\text{penalty}(d) = \begin{cases} 0 & \text{if } d \geq GUARANTEE \\ BASE * \dfrac{GUARANTEE - d}{GUARANTEE} & \text{if } 0 \leq d < GUARANTEE \\ \infty & \text{if } d < 0 \end{cases}$$



Other Language Features

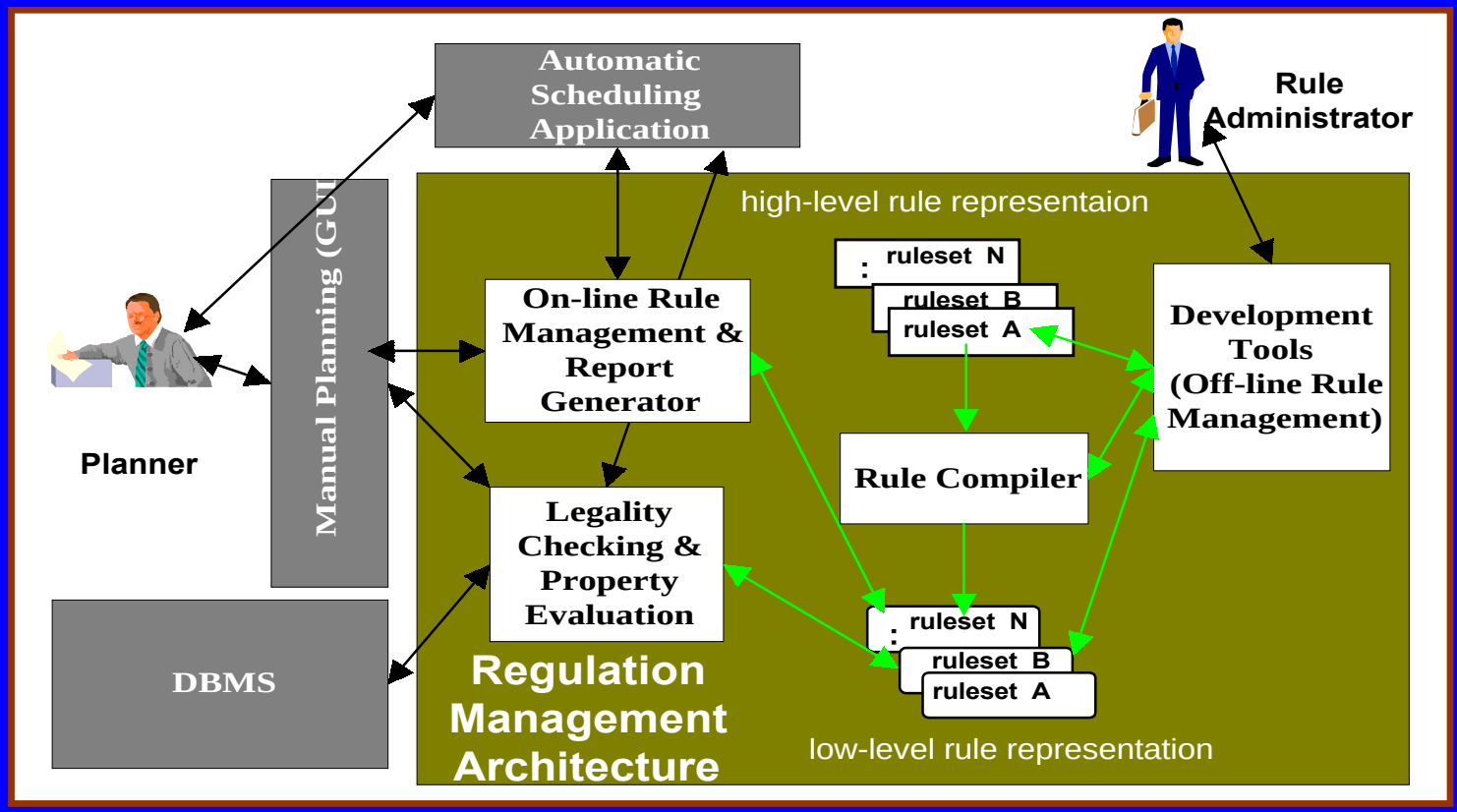
- User-defined functions
- Timewindows based on time groups
- Switch statement
- Rule grouping
- Rule checking with penalties
- Rule checking with priorities

COMPLETE REGULATION MANAGEMENT

Requirements

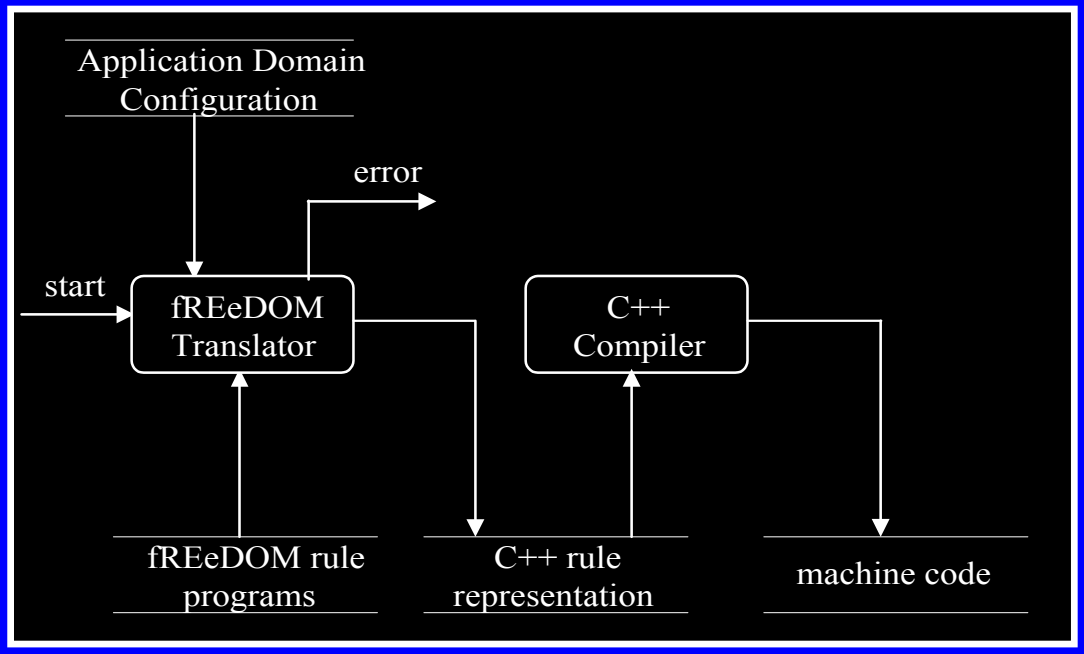
- Regulation modeling language
- Development tools (GUI, editor, compiler, debugger, static analyzer)
- Legality checking and attribute evaluation services
- Report Generator
- On-line rule management (legality checking policies)
- Easy integration with scheduling applications

fREeDOM Regulation Management



fREeDOM Rule Compiler

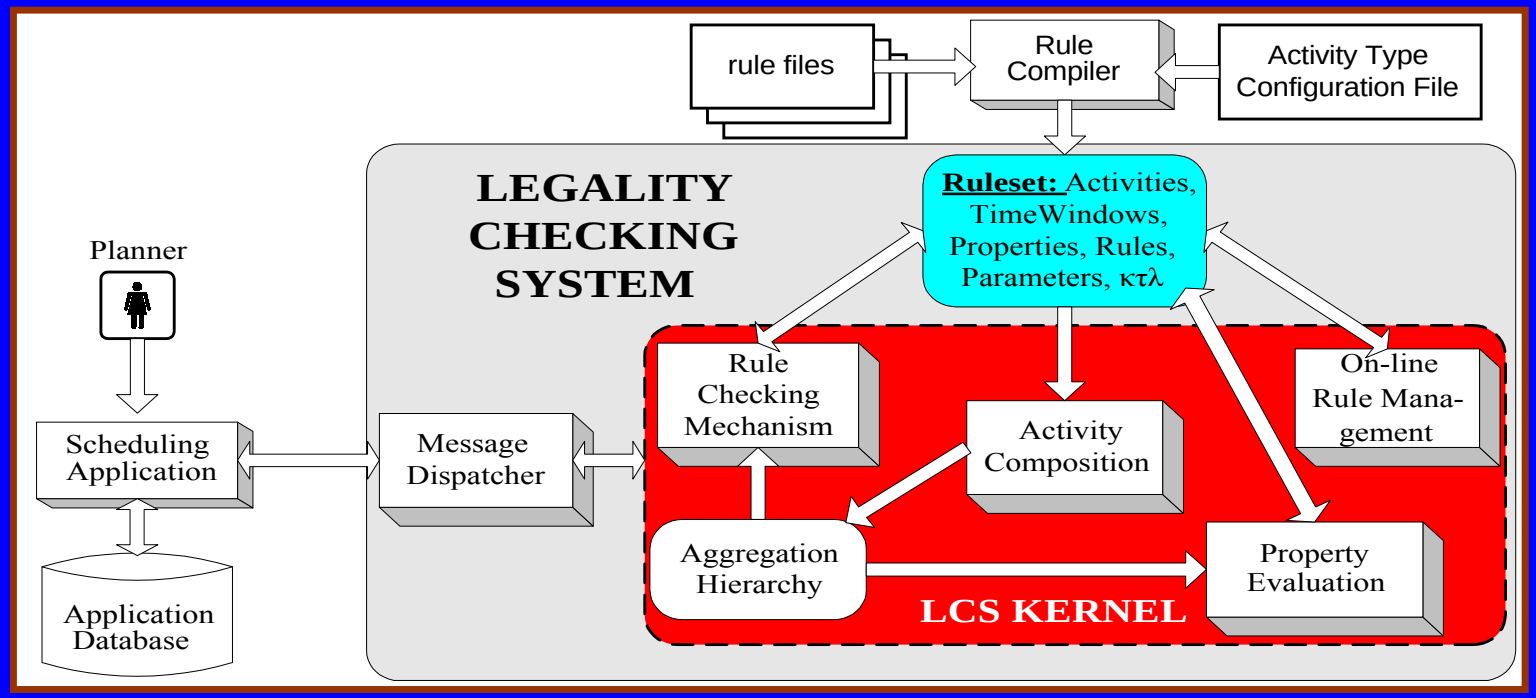
two step compilation



Translation to intermediate C++ classes provides:

- Portability
- Optimized machine code

fREeDOM Legality Checking System



LCS Key Features

- Activity composition (aggregation hierarchy, composition rules)
- Property evaluation (property evaluation rules)
- Rule checking (property constraint rules)
- On-line rule management (enable/disable rules, change parameter values)

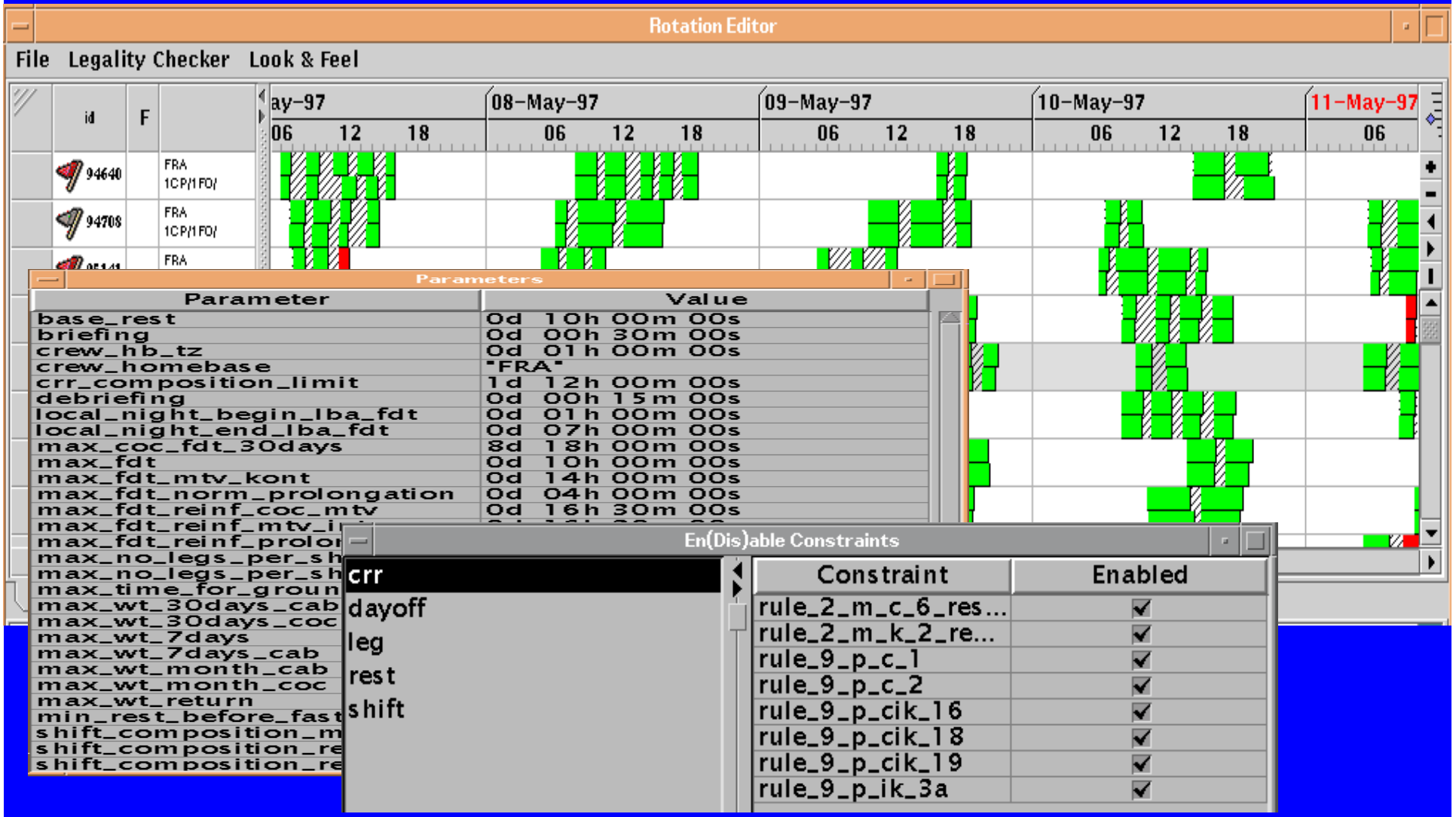
Integrating fREeDOM with Applications

- C++ API
- Abstract data access mechanism (independence from the database scheme of the client system)

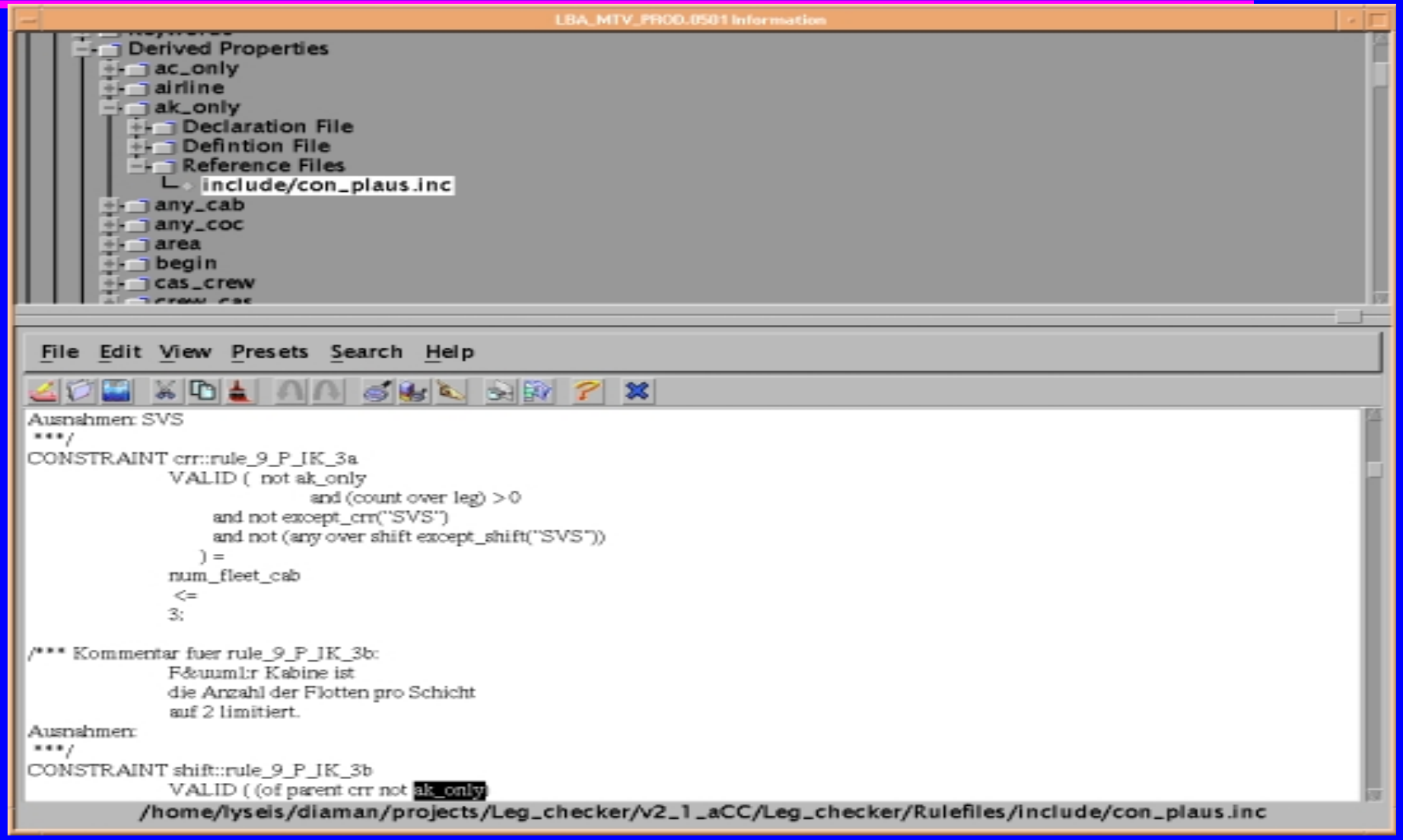
C++ API

- C++ class methods provide system services to client applications
  - **legChecker::openLow()** – creates a line-of-work (low) to host later the activities supplied by the client application
  - **legChecker::addActivity()** – adds an activity to a specific low. Returns to the client application the generated aggregation hierarchy.
  - **legChecker::removeActivity()** – removes an activity from a low. It is useful when the client application applies chronological backtracking
  - **legChecker::checkLegality()** – checks the legality of a low according to the current ruleset
  - **legChecker::closeLow()** – Removes a low
  - **legChecker::getProperty()** – Gets the value of a derived property.
  - **onLineRuleManager::updateDataPart()** – updates on line a rule parameter

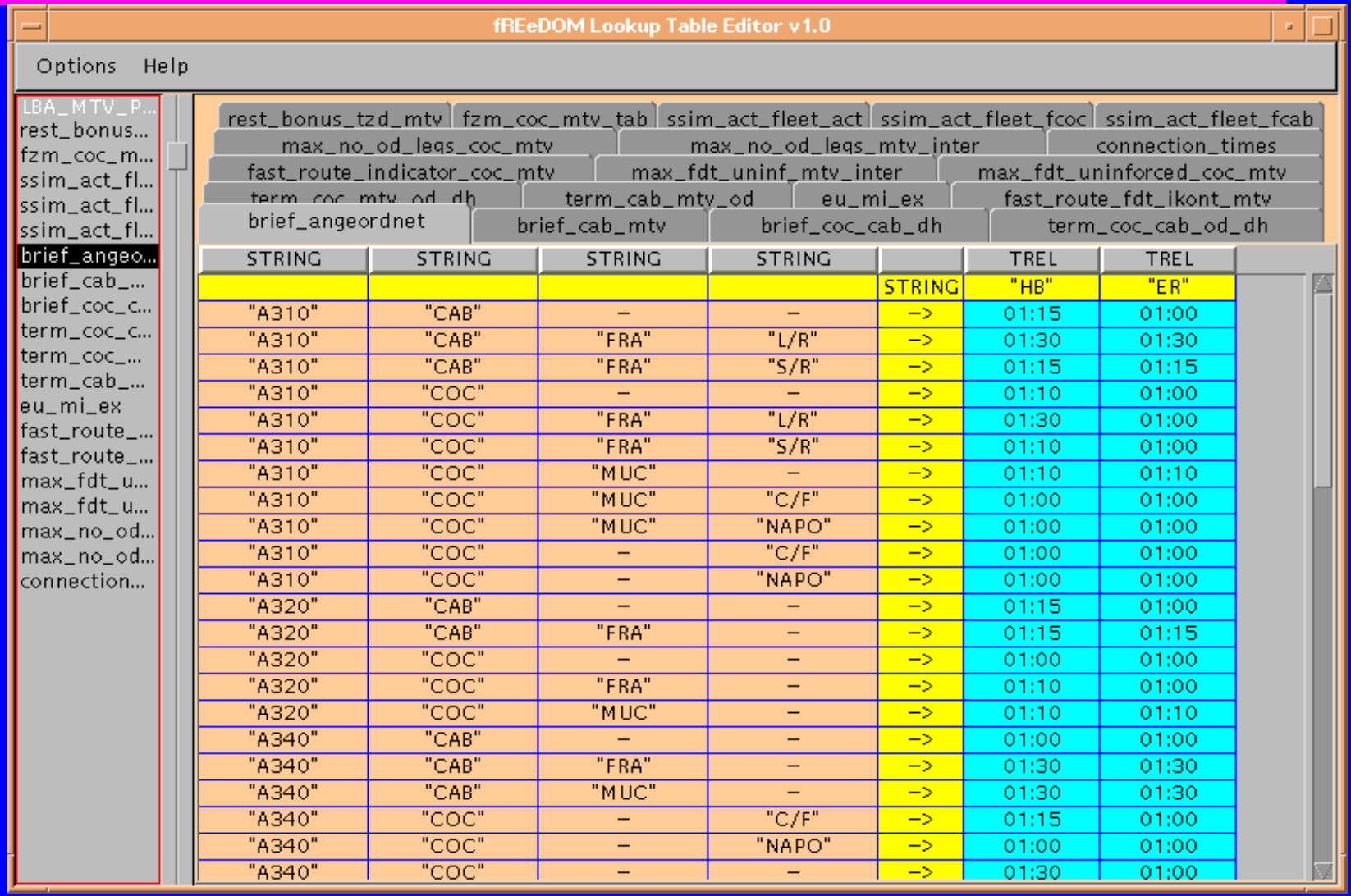
GUI Tool


Rotation Editor
File
Legality Checker
Look & Feel
id
F
08-May-97
09-May-97
10-May-97
11-May-97
06
12
18
94640
FRA
1CP/1FO/
94708
FRA
1CP/1FO/
Parameters
Parameter
Value
base_rest
0d 10h 00m 00s
briefing
0d 00h 30m 00s
crew_hb_tz
0d 01h 00m 00s
crew_homebase
"FRA"
crr_composition_limit
1d 12h 00m 00s
debriefing
0d 00h 15m 00s
local_night_begin_lba_fdt
0d 01h 00m 00s
local_night_end_lba_fdt
0d 07h 00m 00s
max_coc_fdt_30days
8d 18h 00m 00s
max_fdt
0d 10h 00m 00s
max_fdt_mtv_kont
0d 14h 00m 00s
max_fdt_norm_prolongation
0d 04h 00m 00s
max_fdt_reinf_coc_mtv
0d 16h 30m 00s
max_wt_30days_cab
max_wt_30days_coc
max_wt_7days
max_wt_7days_cab
max_wt_month_cab
max_wt_month_coc
max_wt_return
En(Dis)able Constraints
crr
dayoff
leg
rest
shift
Constraint
Enabled
rule_2_m_c_6_res...
rule_2_m_k_2_re...
rule_9_p_c_1
rule_9_p_c_2
rule_9_p_cik_16
rule_9_p_cik_18
rule_9_p_cik_19
rule_9_p_ik_3a

Static Analyzer



Lookup Table Editor



STRING	STRING	STRING	STRING		TREL	TREL
				STRING	"HB"	"ER"
"A310"	"CAB"	–	–	–>	01:15	01:00
"A310"	"CAB"	"FRA"	"L/R"	–>	01:30	01:30
"A310"	"CAB"	"FRA"	"S/R"	–>	01:15	01:15
"A310"	"COC"	–	–	–>	01:10	01:00
"A310"	"COC"	"FRA"	"L/R"	–>	01:30	01:00
"A310"	"COC"	"FRA"	"S/R"	–>	01:10	01:00
"A310"	"COC"	"MUC"	–	–>	01:10	01:10
"A310"	"COC"	"MUC"	"C/F"	–>	01:00	01:00
"A310"	"COC"	"MUC"	"NAPO"	–>	01:00	01:00
"A310"	"COC"	–	"C/F"	–>	01:00	01:00
"A310"	"COC"	–	"NAPO"	–>	01:00	01:00
"A320"	"CAB"	–	–	–>	01:15	01:00
"A320"	"CAB"	"FRA"	–	–>	01:15	01:15
"A320"	"COC"	–	–	–>	01:00	01:00
"A320"	"COC"	"FRA"	–	–>	01:10	01:00
"A320"	"COC"	"MUC"	–	–>	01:10	01:10
"A340"	"CAB"	–	–	–>	01:00	01:00
"A340"	"CAB"	"FRA"	–	–>	01:30	01:30
"A340"	"CAB"	"MUC"	–	–>	01:30	01:30
"A340"	"COC"	–	"C/F"	–>	01:15	01:00
"A340"	"COC"	–	"NAPO"	–>	01:00	01:00
"A340"	"COC"	–	–	–>	01:30	01:00

fREeDOM Applications

- **It is used with great success in the day-to-day crew rescheduling system (DAYSY) of LUFTHANSA Airlines, since 1998**
- **Integrated with a Constraint Logic Programming system (CHIP) for solving the day-to-day resource rescheduling problem**
- **Integrated with a Column Generation based application for solving the crew scheduling problem**
- **Integrated with an efficient pairing generation application for solving the airline crew pairing problem**

CONCLUSIONS

fREeDOM Key Features & Benefits

- Includes a regulation modeling language
  - Least effort to express and extend the rules
- Based on a generic meta-model
- Development tools
  - GUI, compiler, debugger, static analyzer
- On-line management of rule parameters
  - Test different scenarios
- Easy integration to any resource management system
  - C++ API
- Reliable and efficient
  - DAYSY system of LUFTHANSA Airlines, since 1998

Epilogue

- Solving resource scheduling problems in the transportation domain requires applications that need to take into account all the time a large number of regulations
- fREeDOM provides an enterprise with a robust efficient regulation management system for numerous applications that is easy to maintain

Contact

- www.lyseis.gr
- mailto:
  - C.Goumopoulos@lyseis.gr
  - info@lyseis.gr