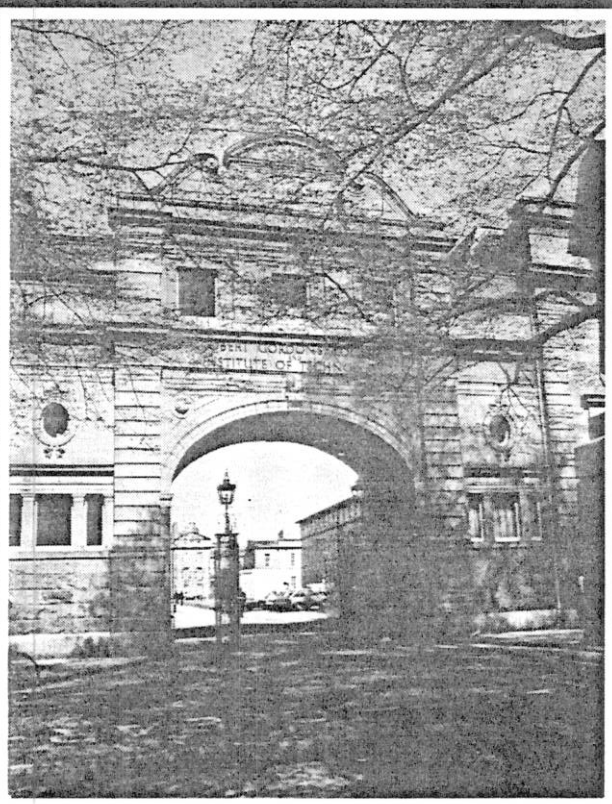

50.

Proceedings of the 25th Universities Power Engineering Conference

held at

Robert Gordon's Institute of Technology, Aberdeen

12th, 13th and 14th September 1990

VOLUME ONE

Open-air substations layout design
A fully adaptable expert system approach.

J.L. Lilien and M. Pallage

Institut d'Electricité Montefiore Université de Liège , Belgium

INTRODUCTION

The whole layout design of an open-air high-voltage (from 72.5 kV to 765 kV) substation has always been a real challenge to systems designers, so true is it that it requires them to be acquainted with numerous fields such as electrical and mechanical engineering.

Adding the strongly parametrizable and country-dependant type of expert appraisement to the fact that very few people can control the entire design strengthens the difficulty in dealing with it.

In connection with industrial partners, we felt the need to build a software whose aim would be to mix this world-spread expertise together with very up-to-date computation codes, database systems and artificial intelligence techniques.

This ambitious work began in September 1988 with the collaboration of an electric devices manufacturer and a network operating staff, namely Merlin Gérin (France) and EDP (Portugal).

Our objectives are twofold:

- providing a reliable and adaptable expert system to be used in the industrial world, by electric devices manufacturers, by power supply operating staff and by engineering offices
- adding a didactic module, which we plan to be a multimedia system, in order to fulfil educational requirements both for our students and for continuous educational training in engineering offices.

DESIGNING AN OPEN-AIR HIGH-VOLTAGE SUBSTATION LAYOUT

We outline in this section the difficulties encountered by system designers when dealing with power systems layout in general and open-air high-voltage substation in particular. We also present the main lines of this design and insist on some of the originalities we did introduce in the expertise.

The complexity of the whole design is of common knowledge for the following reasons:

- system designers have to deal with numerous fields going from electrical engineering to mechanics, some of those fields requiring them to become acquainted with very up-to-date research results
- some of the values to consider, such as security distances often differ from one country to another
- experts for the entire design are very seldom
- like any other design, the high-voltage open-air substation design is not a sequential work and you may have to re-begin a one week engineering office work when noticing that for example the planned structure won't support the electrodynamic stresses due to short-circuits

The design description consists of seven tasks to be fulfilled, each of them taking into account the planification requirements. We intentionally limit the description below to the first five ones, the ground network computation and lightning protection being not yet implemented.

- *TASK 1 - The single-phase diagram choice*: This choice among 15 possibilities is based upon six questions and two criteria (space and cost). The aim of those questions is to determine what we call the *security and flexibility levels*. The security level is defined as follows: In case of busbar or line fault, you have to be able to keep the largest number of devices busy and so use the least number of breakers. The security level expresses this ability. The flexibility level expresses the easiness to deal with maintenance and changes made to the electrical configuration of the substation. One of those questions is for instance: *Do we accept loosing a switch-bay during the maintenance of one breaker?* If the answer is 'no' we can then eliminate all diagrams with one, two or three busbars, disconnectable or not, having only one single breaker per bay and consider the remaining ones like diagrams with one and a half or two breakers per bay or like "loop" diagrams.

  As an example of this expertise, here are some of the characteristics of two diagrams:

  - *disconnectable single busbar diagram with one breaker* : relative needed space: 85, relative cost: 85, number of nodes: 2, in case of busbar fault all connected switch-bays will be out of service, in case of maintenance on one breaker the switch-bay will be out of service, generally used for voltage up to 245 kV
  - *two busbars diagram with one breaker* : relative needed space: 100, relative cost: 100, number of nodes: 2, in case of busbar fault all connected switch-bays will be kept in service after reconnection, in case of maintenance on one breaker the switch-bay will be lost, generally used for voltage up to 765 kV

- *TASK 2 - Electric devices choice*: The difficulty of this task is to select the best fit device (circuit breaker, surge arrester, isolator, current transformer, etc) among a huge number of them. It is one of the longest tasks. To facilitate those choices, we can restrict the set of devices to explore with empirical rules such as: *If available space is restricted, do not consider transverse opening disconnectors (double-rotation disconnectors f.i.)* or *If the minimal temperature at the substation location can be less than -20 degrees centigrade then the family of disconnectors to consider is the pantograph one.*

- *TASK 3 - Busbar type choice and general busbar and switch-bays disposition*: Expertise about this task is very hard to synthesize, the experts studying each case separately without general structure. First the busbar type (either rigid or flexible) is to be chosen using rules such as: *prefer flexible busbars if building a substation in a seismic area* or *beware of flexible busbars if the short-circuit current is high.* After determination of the number of levels (two or three) and insulating distances, the chosen devices are disposed on switch-bays, starting from the overhead line arrival to the busbars and respecting those security distances. The overall needed space is then computed taking the different types of bays, the connecting lines, etc, into account.

- *TASK 4 - Busbar sizing*: It consists of computing the minimal busbar dimensions (external-diameter, section and thickness if rigid busbar, section, sub-conductors number and bundle geometry if flexible busbar) for it to support both rated and short-circuit currents (adiabatic heating) and corona effect. A busbar is then selected in the catalogs using those dimensions. For rigid busbars will follow a sag verification, a frequencies analysis and an aeolian vibrations analysis, each of them making use of methods presented by the IEEE (3). If the busbar is flexible then a sag analysis and a non-resonance verification will be performed using rules such as: *the pendular oscilla-*

tion frequency must be sufficiently away from half the vertical oscillation frequency of the cable.

- *TASK 5 - Computation of static and dynamic overloads*: This is the most important task because it may cause us to reconsider everything that preceedes. It mainly consists of a static and dynamic sizing of the structure based upon the presence of wind, ice, short-circuit current and combinations of them. While the static sizing may generally easily be done using simplified methods, the dynamic behaviour requires much more sophisticated techniques such as the ones recommended by the CIGRE (1,2). For those methods, only the most constraining cases will be studied. For instance, in case of short-circuit on a flexible busbar, the following case will be considered: two-phase isolated fault. Many other values will also be computed such as the pitching stresses for bundle configuration, especially the spacer compression.

WHAT ARE THE CONSTRAINTS WE HAVE TO FACE IF WILLING TO COMPUTERIZE THE DESIGN?

To satisfy the already presented objectives of our system, we have to consider the following constraints:

1. As the target user audience is mainly composed of engineers and students, supposed not to be acquainted with computer science, the first constraint is obviously a **convivial man-machine interface**.
2. If the sequence of tasks to be performed is commonly accepted, the way to achieve those tasks objectives often differ from one country or society to another. Furthermore, the expert appraisement is quite parametrizable, thus providing the following constraint: **providing an easy way to access and modify both expertise and parameters.**
3. The last constraint is to provide the user **explanations** about what is going on.

Our main will is to come to a system where **everything** will be available to the user in a very simple and convivial way. This means that all **formulas, rules, parameters** and **pieces of expertise** should be **accessed** and possibly **modified** by the user (entitled to do it) willing to **fashion** the system his way. As will be presented later, this constraint motivates the adoption of an original control structure.

BUILDING THE PRELIMINARY VERSION

We will in this section present the preliminary version of the system. For reasons explained later we chose to build our system using an object-oriented architecture and of course an object-oriented language. You will for this reason first find a very brief approach of what is the object-oriented paradigm and what are its advantages.

The object-oriented paradigm

Object-oriented programming is a programming style based on the **encapsulation** of both data (the information to handle) and procedure (the way to handle this information) concepts. It is in opposition with the "classical" programming style which maintains a clear gap between data and procedures. The task of program writing thus consists of the definition of a world of independant **objects**, communicating with one another through **messages**, each object being made of a certain quantity of information and procedures to process that information. For instance if we want to compute the permissible current rating of a given busbar:

- in "classical" programming, we will write a procedure computing this value for any busbar, procedure that will receive the given busbar characteristics as arguments
- in object-oriented programming, we will create a busbar object with the characteristics of the given one and ask that object, through a message, to compute its permissible current rating and to return the result.

Objects are the unique type of entities that can be handled by an **object-oriented system**. It means that **everything**, an integer as well as a modeled disconnector, has to be represented as an object. An object is made up of three parts:

- it has a **private memory** consisting of a collection of **attributes** whose value defines its **state**. The value of an attribute is in turn an object. We can distinguish between two types of objects: **complex objects** whose attributes are means of referencing other objects and **primitive objects** that do not have attributes but only a value which is the object itself (an integer for instance)
- it has a collection of **methods** that capture its behaviour. The methods of an object are the only procedures able to manipulate the object private memory and to return its state
- objects communicate with one another by sending messages requiring the receiver object to execute one of its own methods. An object can thus be considered as an independant entity, except for a collection of messages, coming from other objects or itself, it is supposed able to interprete. This collection of messages is called the **communication interface**.

Each object is defined as being part of a **class**. A class describes the structure of a collection of objects having the same attributes and methods, those objects, called **instances**, only differing by the value associated with their attributes. When a message is sent to an instance, the method implementing the response to it is found in the class definition.

Each created class is considered to be **subclass** of an already existing one called its **superclass** and **inherits** the methods and attributes of its superclasses. There is a special class, superclass of all others, serving as root for this hierarchical structure. A new class can bring new attributes and methods adding them to the inherited ones.

The advantages of such a way of programming are well identified:

- the message passing mechanism is such that the object can be seen as a black box responding to a finite number of activations. A program can thus be constructed as a collection of **modules** interacting only through the communication interface, without any idea of the "internal implementation" of one another
- object-orientedness insures **integrity**, that is when two different classes each own a different method with the same name, the correct one will always be activated by the correct receiver. It allows the programmer to get rid of the data types management. For example, the classes *integer*, *fraction* or *float* have their own version of the "+" method and the system will determine which of these versions to activate when meeting a form "[a + b]", depending on the type of the receiver a
- the inheritance mechanism allows the sharing of knowledge between related classes of objects without having to recopy it
- an **object identity** mechanism allows to distinguish each object from the others thus permitting numerous objects to refer to the same one via that identity. This increases the modeling power, for it is the most rational way of expressing for instance that two persons have the same child
- the encapsulation of the data structure and methods applying to it allows a quicker program updating whenever it is decided to modify this data structure

To end this quite abstract section, we illustrate that theory with an example. Suppose you want to modelize busbars. You first have to take into account the fact that there are two kinds of busbars, rigid and flexible ones with specific characteristics for each. There are thus two specializations of the busbar concept, in fact two subclasses of the *busbars* class. Thanks to the inheritance mechanism, you will have to introduce in the *rigid-busbars* and *flexible-busbars* subclasses, only those characteristics strictly specific to them. Among others, the *rigid-busbars* subclass will contain a *wall-thickness* attribute and a method to compute the permissible current rating, while the *flexible-busbars* subclass will have a *number-of-sub-conductors* at-

tribute and, since the computation is different from the one for rigid busbars, a method to compute the permissible current rating (it may be given the same name as the other one). The *busbars* class will be structured with all attributes and methods common to both types of busbars, such as *chosen-material*, *cross-section* and *external-diameter*.

SAPHO, a preliminary version for our system

We started working on SAPHO (stands for Système d'Aide à la conception de Postes à Haute tension Ouverts) in January 1989. What first appeared after a good part of the expertise had been collected was that there was no matter to build an expert system in the computer science meaning of the word. In fact we were facing a very important algorithm making use of lots of techniques, with a lot of unavoidable tasks to perform, the way to achieve them only being subject to changes (mostly due to "historical use" and geographical location). Furthermore, this algorithm was quite sequential although there may occur a backtrack to a previous stage of the design from time to time. SAPHO was nevertheless called "expert system" because it was based on expertise and even if it didn't make use of decision trees or inferences, its aim was still to try behaving like an expert in the field.

The goals of this work are to show the feasibility of such a system, to solve major problems such as interfacing our system with big Fortran codes and try to satisfy the presented constraints. To those ends, it was decided that the six steps of the design would be partially implemented to form a minimal version of the system taking the major part of the problems into account.

We use SPOKE as software support for developing SAPHO. The reasons for this choice are:

- The object-oriented paradigm is particularly well-suited for an easy modelization of physical world components, like a substation or a potential transformer, because it allows the programmer to model the real complexity of such entities without having to simplify them. Another reason for which the object-orientation was chosen is that the design of a man-machine interface is considerably simplified with its use.
- SPOKE is build upon Sun Common Lisp, thus offering an interface to Fortran provided by this language.

The hardware support for our work is a SUN 3/60 workstation with 12 Mbytes RAM.

SAPHO is composed of five modules: a knowledge base, a man-machine interface, a fortran codes interface, an electric devices database and a fully adaptable control structure. As the aim of this paper is to present the adaptability of our system, we will only focuse on the last item.

A fully adaptable control structure. The identified constraints egged us to build a fully adaptable control structure which we ill introduce in this section.

We are confronted with the problem of implementing huge conception tasks, each of them requiring different techniques and expertises. As said above, the corresponding algorithm will be a task chaining one with possible backtracks to previous stages.

We adopted an architecture, based on the following ideas:

- why not consider *task objects* corresponding to elementary operations? We would then have to consider independant entities which we will call actors, each of them having the responsibility of an elementary task such as initialization, computation, help providing or results displaying
- one such actor will have at least two methods, one for its operation and one to activate the next actor once its operation over. It will also own a direct reference to the next actor it has to activate
- there will be a class for each type of actor, and each particular actor of that type will be one of its instances. For example, we can have a class for all actors whose operation is to let the user enter a value for a system variable [1] and instances of that class for each variable to be given a value
- the control structure will then be made of a static network of actors of which the leftmost one is to be activated to start running the system
- one such approach satisfies one of our constraints which is **explanations generating**. Indeed, as each actor is devoted to a type of operation, it is easy to generate rudimentary but sufficient explanations at the activation of any actor and after the result of its operation is obtained. To manage this we can have a standard *trace* for each type of operation and adapt this trace, basing ourselves on the context of the activated actor. For example, if we have an actor devoted to the initialization of the highest-voltage, the (silly in that case) trace will take into account the variable it has to initialize, that is the highest-voltage

This structure cannot however convivially satisfy the remaining constraint that is **access and modification of expertise** since the user would have to deal with generating new actors and modifying bindings. Thus we decided to consider three abstraction levels:

1. **the basic level**, which is made of the already introduced actors network
2. **the middle level**, which is made of a collection of instructions to generate actors and establish bindings between them, and which generates the *basic level* by simply executing those instructions
3. **the electric strategy level**, which allows the user to read and possibly modify the expertise by adding, suppressing, interchanging or modifying high-level instructions, expressed in a very simple formalism. This level generates the *middle level* by execution of those instructions

Our main will is to allow **everything** in our system to be accessed and modified by the user. All system entities such as formulas will thus be available and the user will be able to define his own entities in order to fashion the system his way.

How will he do that? Those entities will be stored as named objects and those names will be used to compose the high-level instructions. The user defining his own entities will give them a name. Suppose for instance that we have a high-level instruction to compute the busbar external diameter to support the rated current using a formula entity named *formula_1*. If the user whishes to use another formula, he will only have to enter it in the system, in a way described later, to give it a name and to replace *formula_1* by that name in the instruction.

All entities and expertise (the high level instructions) will be editable in a suited editor, allowing the user to modify them or to compose his own ones by simply selecting on menus everything he needs. If we consider formulas, those menus will present mathematical operators and all system variables (their name). A menu for all system entities (their name) will always be user available in order for him to access these entities or to select their name for insertion into a high-level instruction. The user willing to compose a new high level instruction will be provided a menu with the simple syntax of those instructions and the menus for all system entities.

The system entities are:

- **formulas**: a formula will be written in a lisp-like formalism for the simplest of them. For complex computations, FORTRAN codes will be used as formulas, assuming the user is familiar to this language
- **parameters**: a collection of parameters will be available for value modification, that is the user will be able to give a value corresponding to his country to parameters such as *the percentage of the span to consider in order to proceed to the busbar sag analysis*. He will also be allowed to add new parameters which he will insert in his formulas
- **system variables**: in order to be used by high-level instructions and formulas, the system variables such as the highest

[1] not such an easy stuff as it seems, because, this type of actor will have to generate a menu with all possible values, allow the user to ask for help about the variable and allow him to enter his value in a convivial way whenever he is not satisfied with the proposed ones

voltage will be available by menu. Each variable is assigned a collection or interval of possible values and the user will be abilited to modify them

- **selection criteria**: the attributes of all classes of the knowledge base will always be available to compose the selection high-level instructions
- **rules**: a collection of rules will also be user available and definable
- **checkings**: a checking is a collection of tests to validate the value assigned to a system variable with regard to already initialized ones. They are to ensure a consistency of the system. For instance, a checking based on the highest voltage can be done when initializing the rated current

It is now time to introduce the high level instructions. There will only be three types of them because our experience with the expertise told us it would be sufficient [2]. Each of those instructions will correspond to at least one actor, selection operations for instance requiring many of them. We will only give one example of the most complex instructions for each type:

1. **initializations** : For initialization of the planified highest voltage and basic insulation level, we use the following instruction:

```
[sapho init_and_test highest_voltage
 using (< 300)
  (if_ok [init lightning_impulse])
  (if_not_ok [init switching_impulse])]
```

That particular instruction means that once a value has been given to the highest voltage, a test on this value is made in order to determine the next instruction to execute. That is, if the highest voltage is less than 300 kV then proceed to the lightning impulse initialization, else give a value to the switching impulse.

2. **computations** :

```
[sapho test_then_compute external_diameter
 using (= busbar_type rigid)
  (if_ok with formula_1)
  (if_not_ok with formula_2)]
```

In this case, the formula to be applied in order to compute the busbar external diameter depends on the busbar type.

3. **selections** :

```
[sapho select_single disconnector
 using criteria
  ((= disc_type chosen_disc_type 1)
  (= manufacturer (siemens merlin_gerin) 1)
  (>= disc_highest_voltage highest_voltage
          1 25%)
  (>= disc_bil bil 1)
  (>= disc_rated_current rated_current 0.5)
  (>= disc_sh_circuit_current
                  sh_circuit_current 0.5))]
```

This example asks the system to select a single device for each of the mentioned manufacturers, device that will be of the chosen type and that will satisfy all specified criteria. It is clear that a solution will always have to be provided, so, if a criterion cannot be satisfied by the stored devices, the system will have to proceed to an approximation on that criterion, that is, if there are devices satisfying all criteria but the one involving bil, the selection will return among those devices, the one whose disc_bil attribute is the closest to the bil. In that rare case, the system will go on working with that approximation if told to, telling you anyway to contact a manufacturer to build a special device for your needs. If you prefer waiting for the dimensions of that special device, the session will be stopped until you enter the new device in the system.

[2] for all tasks but the *disposition* one that will need the integration of a CAD tool in the system and for which other instructions will be introduced

The numbers at the end of each criterion express a relative importance among those criteria. This will be useful to determine a single device. In effect, suppose we have a collection of possible devices, if we wish to select the most adequate one, we have to sort those devices on the criteria, in order to get an ordering of devices. If we consider the example, it can easily be seen that the device satisfying all equality criteria and whose attributes involved in inequality criteria are the smallest ones to be found in the ordering, will be selected. What if we are to decide between two devices having the following characteristics:

- A1: supported highest voltage 245, supported bil 1050, supported rated current 1600 and supported short-circuit current 40
- A2: supported highest voltage 300, supported bil 1175, supported rated current 1200 and supported short-circuit current 31

It must be possible to tell the system to begin sorting for instance on the highest voltage, then on the bil and so on. An hierarchy for criteria is thus to be established for the final sorting.

Some criteria are to be considered carefully. If we consider for example the highest voltage, it is not acceptable to select devices supporting 765kV if the planified highest voltage is 420kV even if obliged to go that up because of other criteria to satisfy. That is the purpose of the percentage found at the end of those criteria. In our example it tells the system not to go beyond 25% of the planified highest voltage.

We hope that the syntax of those instructions will not be too discouraging so that the control structure will effectly be adaptable. The instructions are of so high level of abstraction that we believe the entire expertise can be expressed using few of them thus insuring an easy reading and understanding of it.

We design SAPHO in order to satisfy both students, in a didactic point of view and the industrial world, by the time saving it brings and the facts that it synthesizes the procedure to be followed and that it helps not to forget any detail.

ACKNOWLEDGEMENTS

We would like to thank Mr. J-C Leroy from Merlin Gérin and Mr. F. Mira from EDP for the very valuable help in the expertise collection and synthesis. We would also like to emphasize the initiating procedure of this project made by Mr. M. China from Merlin-Gérin and the help of his A.I. team.

TRADEMARKS

REFERENCES

1. CIGRE Brochure SC 23 (Substations), 1987, *"The mechanical effects of short-circuit currents in open air substations"*.
2. Lehmann, W., Lilien, J.L., and Orkisz J., 1982, *"The mechanical effects of short-circuit currents in substations with flexible conductors - Numerical methods - Computer approach"*, CIGRE report 23-08.
3. IEEE Substation committee, WG 69.1, 1979, *"Guide for design of substation rigid bus structures"*.