A Knowledge-Based Approach

to Integration of Image Processing Procedures

Véronique Clément and Monique Thonnat

INRIA Sophia Antipolis

Running head: Integration of image processing procedures

Address:

Véronique Clément

INRIA Sophia Antipolis

2004, Route des Lucioles

F-06565 Valbonne Cedex France

Tel: (33) 93-65-78-57 Fax: (33) 93-65-77-66

e-mail: vclement@mirsa.inria.fr

Abstract

This paper deals with the integration of image processing procedures. Three kinds of integration are distinguished: physical, syntactical, and semantic integration. The notion of semantic integration of programs is developed here; in this kind of integration, the function of the programs, and how to optimize their utilization are explicited. After a short presentation of several recently developed systems performing semantic integration of programs, we describe how we model knowledge and reasoning in our system named OCAPI. Two examples of use of OCAPI are shown; the first example is the integration of a stereovisual process, the second one is the integration of a system for the morphological description of galaxies. Finally, the model and the mechanisms used in OCAPI are compared to those of the previously presented systems.

List of Symbols

Only the usual arithmetical symbols are used: $+ - * / \% <$

1 Introduction

The integration of a set of existing programs is a very important and difficult task. The major difficulty of the integration phase is that most of the time modules are developed independently, either by different specialists or at different periods of time. Knowledge of use of these modules is not included in the code, but split into several brains or forgotten.

Although this domain is still young, three types of integration (physical, syntactical, and semantic) can be distinguished. More precisely:

- usually integration of programs is understood as the adaptation of the algorithms to a specialized hardware; this kind of integration is what we call **physical integration** of software onto a given architecture;
- integration of a set of programs in a command language or an interface system is what we call **syntactical integration**; it is the integration of the syntactical information needed to run the programs;
- finally a third kind of integration is possible: the integration in a system of knowledge of the goals of the programs, knowledge of conditions under which they are applicable, and knowledge of relations between the programs; we speak of **semantic integration** or intelligent integrated systems.

Usually, image processing systems are composed of a set of programs or subroutines which can be used directly or through a command language. These systems only provide a physical integration of the procedures or at best a syntactical integration; that is the use of abstract commands instead of operating system or programming language syntax. If the set of programs are semantically integrated, we may have an intelligent image processing system with a certain degree of autonomy. Such intelligent image processing systems could facilitate the developing of new image processing applications since certain tasks can be automated (like the selection of the best programs to reach a goal or the initialization of

some parameters); moreover, if their knowledge is sufficiently developed, this can extend the use of image processing systems: they can be directly used by non specialists in image processing (like biologists, astronomers...) or integrated on autonomous systems working in complex and variable environments (like mobile robots or flexible arms). Physical, and syntactical integration of programs for robotics purposes only deals with the problem of **robustness** of the processing (the processing is fixed, but the input data can be slightly noisy); with semantic integration of programs it is also possible to deal with the problem of **flexibility** of the processing (the processing can adapt itself to various conditions).

Various methodologies, generally based on artificial intelligence techniques, have been investigated to develop intelligent image processing systems. So a clear distinction has to be made to discriminate between those which are intelligent algorithms, intelligent interpretation systems, or intelligent integrated systems. A classification can easily be performed according to the nature of the knowledge these systems contain **explicitly**:

- **intelligent algorithms** express in detail *how* programs work, and describe explicitly the *internal mechanisms* of the algorithms. This kind of knowledge is interesting for innovation, generalization or educational purpose. One can mention as examples of such intelligent algorithms some interesting work in segmentation (the system developed by Nazif and Levine [1], LLVE [2], and VISIONS [3]).

- **intelligent interpretation systems** contain explicit knowledge on the *modelling of objects* of the world. As examples of such intelligent interpretation systems, one can mention ACRONYM [4] which uses a geometrical modelling of domain objects, CLASSIC [5] which uses a semantic modelling of classes, and the system developed by Ikeuchi [6] which generates an object recognition program from a 3D CAD model of the object.

- **intelligent integrated systems** contain (or have to contain) all the knowledge needed for the selection and the use of programs seen as black boxes. Some algorithms in the library can of course perform object recognition, and they can even be

intelligent; but they are considered by the integration system like a black box. In such intelligent integrated systems, *what* programs do (their goal) and *when* (under which conditions) are expressed explicitly.

In this paper, we will focus on semantic integration of programs in image processing. Work in this domain is very recent [7] [8] [9] [10] [11] [12]. So, in the next section, we present some systems which are representative ones, either for their state of development or their mechanisms. In section 3 we present the model of reasoning and knowledge used in our system named OCAPI. Then in section 4 we describe two examples of integration of image processing based on OCAPI: one for stereovision, and the other for morphological description of galaxies. In the last section, we compare the models used in the systems we have presented.

2 Recent work in semantic integration

Integration of image processing modules has been recently addressed using intelligent systems. In this section, five such systems are presented; the goals of the authors are quite different: interactive help versus fully automatic system, domain specific tool versus general system, generation of programs or scripts versus execution of commands.

2.1 Expert assistant

In the field of astronomy, Johnston introduces the concept of an expert *Data Analysis Assistant*. In fact, astronomers manipulate image analysis systems which offer numerous functionalities. Depending on how and when the data have been obtained, astronomers often have to use various image analysis systems. These systems are usually implemented as command languages. So, Johnston has proposed Expert assistant [7], a system of generation of commands, independent from the image analysis system. A prototype has been developed using the commercial expert system shell KEE [13]; it addresses the

problem of CCD calibration as a simple test case for its methodology. Two different image analysis systems are available (MIDAS and SDAS/IRAF).

Expert assistant allows the expert in image analysis to represent the knowledge of different kinds of data, instrument modes, and data analysis operations expressed in term of tasks. Tasks can be primitive or compound. Control is done by production rules, and reasons about data properties, merges redundant tasks, and expands compound tasks, but without accessing the image data. In a first step, a symbolic plan is generated. In a second step, this plan is translated into commands for one of the two available image analysis systems.

So this expert system simply generates optimized sequences of commands given a symbolic description of the data.

2.2 Toriu's system

Toriu *et al.* have been interested in the general domain of image processing, without targeting any specific domain of application. They are concerned with improving the productivity of beginners and non specialists who have to develop specific algorithms for an application. So, an expert system [8], working on the FIVIS image processing system, has been implemented in Lisp. This system provides functions that would help users to construct their application, by automatically selecting and combining various image processing modules. It is not just a prototype, but already a commercial tool; 120 image processing modules are available, and 40 image attribute names are used.

Given a predefined goal (such as segmentation, image quality improvement or feature extraction), and a symbolic description of the attributes of the image (image type, noise level, texture...), this system selects a global processing (using inferences), and image processing modules (by matching between the characteristics of the image and those of the modules); it also runs them. So, selection and execution of the modules are done in the same process: a module is selected and immediately executed. A distinction is

made between abstract goal, decomposition into subgoals, and programs. Programs are explicitly described, using frames. Planning is performed by production rules (260) either for the decomposition of the goal into subgoals, or for the selection of the programs. The system can ask the user to qualify the input data (initial or intermediate images) in order to process them; but there is nonevaluation on the quality of the results. This system is highly dependent on the image processing system (FIVIS).

This system offers a conversational help to the user to perform image processing.

2.3 EXPLAIN

The work presented in [9] is also involved in the general domain of image processing, without any specific application field. Tanaka and Sueda developed a system which assists the non-expert in using a package of image processing algorithms to obtain a required image from a given one. As special feature, a knowledge acquisition facility is incorporated; it captures knowledge about image processing procedures through interaction with the domain expert. EXPLAIN has subsystems for consultation, and image processing. They are implemented as a set of PROLOG clauses.

The consultation subsystem interacts with the user to guide image processing, and suggests an appropriate course of actions. The problem is specified using keywords (like enhancement or segmentation), and some approximated properties of the image to be processed (color/black and white, noise level, contrast). The process decomposes the given requirement into a sequence of image processing algorithms, using production rules. This decomposition into sequences, which is considered to be subgoaling, is done using the depth first search strategy; when a subgoal corresponding directly to an actual algorithm, is reached, the image processing subsystem is called. Then these algorithms are run, and their results are displayed to the user; in case of unsatisfactory results, the values of the parameters are modified, or another command is tried.

EXPLAIN is an image processing expert system; it runs the commands on the images, and provides a trial-and-error mechanism to deliver satisfactory results.

2.4 DIA-ES

H.Tamura and K.Sakaue [14] proposed the DIA-ES system (Digital Image Analysis - Expert System). It helps, in a semi automatic way, to design an algorithm for a specific application. It has been used to design algorithms for a problem of pattern recognition [10]. DIA-ES is composed of two subsystems, the semantics and the syntactic processor, which are developed on a symbolics lisp machine (using Uranus prolog/KR).

Given an image and a problem, the semantics processor selects the appropriate processing modules. This is done using knowledge about image processing methods, and interactively with the application designer. An execution tree, built by production rules, describes the processing being elaborated. These rules express the decomposition of the goal into subgoals, at different levels of abstraction, or modifications in the structure of the execution tree (suppression, insertion of new operations). Manipulating the execution tree is a way to perform automatic adjustment of the processing. This tool is highly interactive. Evaluation of results, and adjustment of parameters are performed by the user.

The syntactic processor has been presented by K.Sakaue and H.Tamura in [15]; given a sequence of abstract commands, it generates a program using knowledge of the data, and the algorithm structures of the selected modules; each one is described with its name, and informations concerning its arguments. An application has been developed with the library SPIDER [16].

So, from an image, its characteristics, and a goal, DIA-ES generates sequences of abstract commands (semantics processor), and optimized programs (syntactic processor).

2.5 Bailey's system

Bailey is interested in making easier the development of image processing algorithms for people who are specialists in some application domain but not in image processing. A prototype presented in [11] has been developed, but not yet used for some specific

application. The system is composed of the algorithm generation subsystem implemented in PROLOG, and of an image processing subsystem which is currently a command-based system very similar to VIPS [17]. But the design has been done in such a way that any other image processing could also be used.

In a first step, the system interacts with the user to determine precisely the type of image processing task to perform; then recommendations are made on the way to obtain a representative image to develop the algorithm on. In a second step, an architecture of algorithm is selected among the architectures described in the knowledge base, according to the type of task to perform; this is done using selection rules. Each architecture contains a decomposition of a main task into substeps. For each substep, an operation which can be performed by the image processing subsystem is selected, and performed. It is interesting to observe that initialization of parameters can be done through recursive calls to the system. The results of an operator (not a goal) are evaluated by the user; if they are not satisfactory, another operation or sequence of operations is tried; but there is no loop mechanism to repeat executions of a same operator for the same task. When all the results are satisfactory, the algorithm is provided to the user, and can be tested on other images.

So, this prototype provides computer assisted generation of image processing algorithms.

2.6 Conclusion

We can notice that all the presented systems, have the architecture of an expert system; they are expert systems themselves or generator of expert systems, depending on their generality. In fact, this methodology permits to separate particular knowledge about an application domain and programs, and the control of the reasoning for integration; so the development and the reutilization of such systems are facilitated.

3 The OCAPI system

3.1 Objective

In order to perform semantic integration of programs, we are concerned with the problems of modeling knowledge and reasoning which are used in the development of image processing systems. Due to the necessity for generality, the model has to be independent from any specific application or domain of application, and even from any image processing system. First, we propose a model verifying these requirements; second, we describe OCAPI, an implementation of this model which automates execution of image processing; i.e., given data to be processed, a goal to reach, and eventual constraints on the results, the system generates, performs, and optimizes the processing. Systems mentioned in the previous section do not take into account the same constraints. For instance, Expert assistant cannot run the script of commands that it has generated; Toriu's system is heavily depending on the FIVIS image processing system; and none of those systems procides a fully automatic mode of utilization.

In this section, our model and its implementation are presented. First, a model of the reasoning performed in semantic integration is described (the types of reasoning, and the various techniques). Second, a model of knowledge involved in semantic integration of programs is presented (typology of various concepts, and relations between them). Then, we conclude on the implementation of the OCAPI system. Some examples of integration using OCAPI are given in the following section.

3.2 Model of reasoning

[2] and [18] discuss how to reason during the development of an image processing application. This can be summarized using examples as follows:

-1- selection of substeps or subgoals; for example: to reach a certain goal, one has first to perform a segmentation, then to select the biggest region, then to describe its

shape;

-2- selection of operators, algorithms or methods; for example: for image segmentation three methods can exist while, in a particular context, we choose the operator seg1;

-3- initialization of the input arguments; for example: set threshold t=0.45 and input image=im21;

-4- effective execution of programs ; for example: run the command " seg1 im21 -t 0.45"

-5- test of the results (satisfactory or not); for example: if the size is below 30, the segmentation is too fine;

-6- new execution of programs with other values of parameters; for example: if the segmentation is too fine, increase t by 10%;

-7- if the results are still not correct, selection of other methods; for example: if the operator seg1 fails then use the operator seg2.

Phases 1 and 2 correspond to a **planning** reasoning. In fact, planning is "to describe a set of actions (or a plan) that can be expected to allow the system to reach a desired goal" [19]. Here, the actions are the programs; a plan is a chaining of programs, while goals are functionalities to provide. Phases 3 to 6 execute the generated plan, and control its quality; so these phases correspond to a **control of execution** reasoning. The last phase corresponds to a **replanning** reasoning; i.e., execution of the plan is stopped, and the plan has to be changed; this necessitates a new planning phase.

Two kinds of reasoning need to be developed: planning, and control of execution.

3.2.1 Planning for integration

In this paragraph, the question is: which planning technique is adequate for semantic integration of programs?

In problem solving, planning is to find a plan for achieving a goal according to a given state of the world. A plan is defined as an ordered list of actions to perform. Three main techniques have been developed in planning: non hierarchical, hierarchical and skeletal planning [20]. While the two first methods concentrate on the reduction of conflicts in the plan steps, the last one emphasizes the expertise of the specific domain. The third method, also called script-based planning, assumes that abstract or skeletal plans exist. A skeletal plan contains the basic steps to solve a problem. First, the best skeleton is selected, then refined.

In image processing, we have few sensitive conflicting subgoals; so the two first techniques are not necessary. As we hold knowledge, at several levels of abstraction, of typical processing methods, a skeletal technique providing a refinement mechanism is appropriate. Another point is the handling of real world data; we have to deal with unreliable or uncomplete information; the state of the world (initial, final or even intermediate) can not be described exhaustively. So we have to use a technique based on the description of the actions to be performed, and not on the different possible states of the world. Given these prerequisites, a skeletal planning technique proves to be adequate for the planning phase. A plan can eventually be modified during a replanning phase by replacements or insertions of actions.

3.2.2 Control of execution for integration

We are interested, here, in defining the type of control of execution which is the best adapted for the integration of programs.

Four main types of control of execution [19] have been developed:

- prediction of failures, and planning of additional actions which wait until the world corresponds to what is expected (example: waiting for a light to turn green before crossing a street);
- prediction of failures, and integration in the plan of tests and correcting actions;

- prediction of failures, and integration in the plan of tests and alternatives;
- integration of the planning process, and control of the execution process.

In our context, in image processing, we do not handle secure actions: predicting exactly the behaviour of an image processing operator on an image is not possible. So, a trial-and-error strategy is needed, as well as the interleaving of planning and control of execution processes.

3.2.3 Reasoning for integration

In conclusion, two main tasks have to be performed (Figure 1).

First, a planning phase reasons about an appropriate plan to guide the processing of the given images. This step determines the list of instructions to run according to the request of the user: this is done using a progressive refinement strategy. Second, the control of execution of the plan is done through trial-and-error experiments. The quality of the results obtained from the input images is checked (evaluation of the results), and in case of unsatisfactory results, a mechanism adapts the values of the parameters of the image processing algorithms (adjustment of the parameters). The sequence of operators can also be modified using a replanning mechanism if needed.

3.3 Model of knowledge

In this section, we describe in detail the model of knowledge we have defined for semantic integration of programs. First we define a typology of the different kinds of knowledge we need, then we describe how all these concepts are related and structured.

3.3.1 Typology of the various concepts

In order to model knowledge of integration, and to facilitate the building of knowledge bases, the important concepts have to be found.

First of all, when we process an image, we always have in mind an objective or a goal to reach. This notion of goal is close to that of task we need to serve. In the following section, we use the term *goal* which is defined below:

> a **goal** represents an image processing functionality. This functionality or objective can be reached by a program or a complex processing. For instance, *to smooth an image* is a goal; in the same way, *to count objects* in an image is another one.

We can note that programs in a library implement image processing functionalities or tasks to perform, but they **are not** image processing functionalities. In fact, we distinguish this abstract notion of *goal* from that of available *operators:*

> **operators** are actions which can be performed. They contain specific knowledge to solve a given goal. An operator can be a particular program (it will be referred to as **primitive operator**) or a particular sequence of processings (it will be referred to as **complex operator**). For instance, the primitive operator *median3* is a program implementing a median filter with window 3×3, while the complex operator *count-objects1* is a succession of several processings (smoothing, segmentation, sorting, enumeration).

In addition to these concepts, two other notions must be introduced: notion of context and that of request. First, in order to find the best sequences of programs, the *context* has to be taken into account:

> the **context** is a description of the input data, their conditions of acquisition, the application domain, and even semantic information on the supposed contents of the scene. For instance, a context can be: *c-astro: the application domain is astronomy, the optical instrument is a high quality telescope, the digitizer is a microdensitometer, the scenes are extended sources scenes, and stars are present.* Another example would be *c-robot-in: the application domain is robotics, the optical instrument is a CCD camera, the digitizer is a standard one, scenes are indoor scenes.*

Second, a problem in vision is expressed by its goal (or functionality), but also by the data to be processed, and eventual constraints on the results to be obtained. On the same

way we can differentiate instances and classes, a request can be understood as a particular case of a generic goal. The term of **request** will denote the following information:

> a **request** states which goal has to be reached, the data of the particular case to work on (for example, the name of the input image), the required quality for the results, and the context. A request is thus a query to solve a goal for a specific case. For instance, a request can be *r-cont: contours detection of the image "view1"*; another one can be *r-region: region segmentation of the image "view2", the size of the regions must be greater than 2500 pixels, and the context is "robot-in".*

Operators (primitive or complex), and goals work on a list of input and output arguments as programs do. For instance, the goal *smoothing* has two arguments: one input argument (the input image), and one output argument (the smoothed image); the primitive operator *median1* implements a way to reach the goal *smoothing*, and has three arguments: two input arguments (the input image, and the window size), and one output argument (the smoothed image). We can distinguish two classes among the arguments, *data* and *parameters:*

> **data arguments** have fixed values, which are set (input data) or computed (results). In the previous example, *input image* and *smoothed image* were data arguments. Often, operators performing the same goal have the same data arguments;
>
> numerous image processing operators have tunable parameters. They will be referred to as **parameters arguments**; they have adjustable values and are always input arguments. In the previous example, *window size* was a parameter argument. The description of these arguments is important in the knowledge base; in order to optimize the processing, we not only need to know their value (e.g.: *window size = 3*) but also their range (*minimum = 3, maximum = 11*), an eventual default value (*default = 3*), and their type (*integer* or *real*). These arguments, and their characteristics are specific to each operator.

We have seen in the previous section concerning the model of reasoning that we have several phases in integration: how to choose among methods, how to initialize input argu-

ments, how to execute programs, how to evaluate results, and eventually how to modify the processing with the determination of new input values for programs or selection of other programs. From this description, we can distinguish four notions: choice of operators, evaluation of results, initialization of parameters, and adjustment of parameters:

by the notion of **choice**, we mean the knowledge of how to select, among all the available operators, the operator(s) which is (are) the most pertinent, according to the data, and the context. For example: *if the images have a high contrast, use a contour based segmentation operator.*

by the notion of **evaluation**, we mean the knowledge of how to assess the results provided by the selected operator after its execution, taking into account constraints expressed in the request. For example: *if the number of the segmented regions is greater than 100, the segmentation is too fine.*

by the notion of **initialization**, we mean the knowledge of how to initialize values of input arguments. Various mechanisms are required, as default values or computation method depending on the values of other arguments. For example: *for the parameter window size, the default value is 3* .

by the notion of **adjustment**, we mean the knowledge of how to modify values of parameters after a negative evaluation. For example: *if the segmentation is too fine, increase the value of the threshold th-sigma.*

3.3.2 Relationship between these concepts

In the previous paragraph, we have defined several concepts (goal, primitive or complex operator, context, request, data or parameter arguments, choice, evaluation, initialization and adjustment). These notions are not isolated, but are related together.

Relationship between goals, operators and requests: Operators are always related to a precise goal; but the level of abstraction of a goal (abstract notion), and of its

related operators (concrete executable actions) are different. A complex operator implements a set of processings, which is referred to as a **decomposition**. A decomposition is not a set of programs, neither a set of goals, but is a set of requests. To be more precise: a decomposition is not just a set of programs, because, for example, we can mention the case of a particular operator for stereovision which can be decomposed into two steps: extraction of primitives, and matching of primitives; but if several operators (complex or primitive) are able to perform extraction of primitives, such a decomposition does not impose which one to use. In the same way, a decomposition is not just a set of goals; in fact, in a particular decomposition, data flow of input and output arguments have to be managed; for example, the output of extraction of primitives is one input to the matching. To dictate some constraints about the results of each step must also be possible: matching of primitives will be performed only if the number of extracted primitives is significant for the given problem. So, each step of a decomposition is a request to solve a goal in a special context, with restrictions on the input, and eventual specifications on the results.

Relationship between the steps of a decomposition: The set of requests involved in a decomposition can be seen as a tree where the leaves are the requests, and the full nodes express the temporal link between the requests.

Two types of temporal links are used, representing:

- *sequentiality:* Request 1 THEN Request 2.
  For example: filtering THEN contour detection THEN chaining.
- *parallelism:* Request 1 AND Request 2.
  For example, in stereovision: extraction of primitives on the left image AND extraction of primitives on the right image.

The example of Figure 2 corresponds to requests which are related on this way: first Request 1 then Request 2 then [Request 3 and Request 3' and Request 3"] then Request 4.

Relationship between choice, evaluation and goals: We have seen that the available operators to perform a goal are related to it. But, the goal has to know how to select among them, the operator which is the most suitable for a specific problem: the knowl-

edge of how to choose the operator is global to the goal, and not local to each operator. Evaluation of the results is also a mechanism which has to be related to the goal. In fact, the goal knows what results are required: it chooses the way to perform them, and then has to verify their conformity to the requirements. The criteria of evaluation must be common for the same goal, because the evaluation of the results has to be independent from the operator which performs the processing. So a goal can be seen as an item able to decide how to solve a given problem, and to validate the results.

Relationship between initialization, adjustment and operators: An operator has to know how to initialize its parameters. In fact, each operator uses its input parameters in a specific way (meaning, format, range, sensitivity...). *A fortiori,* knowledge about adjustment of the input parameters depends on the operator: for a given bad result, the sensivity of the various parameters depends highly on the particular algorithm and implementation for a program, and on the particular decomposition into subrequests for a complex operator. So these mechanisms must be located on each operator. An operator is an item able to manage its parameters.

3.3.3 Knowledge for integration

We can summarize the model as follows: the user submits a request which relates to a goal, a set of input data, and a specific context. As a goal can be solved by several operators, choice knowledge is used to select an operator among them. An operator can be a program (primitive operator), or can have a proper decomposition into subrequests (complex operator). Before the execution of the operator, initialization knowledge sets the values of its parameters. The results of the execution are estimated by evaluation knowledge. Adjustment knowledge modifies the values of the parameters in case of unsatisfactory results.

3.4 Implementation

This model of reasoning and knowledge for semantic integration of programs has been implemented within an expert system generator architecture. Figure 3 explains the relations between the user, the knowledge base level, the data, and the programs. It shows also the architecture of an expert system built with OCAPI: it is compound with a knowledge base, a base of facts, and the control structures (OCAPI).

OCAPI has been developed in Le_Lisp, [1] and its object oriented facilities.

3.4.1 Implementation of the knowledge model

We have seen that strong relations exist between the various concepts we have introduced. These relations structure the knowledge base in order to facilitate the expression of the knowledge and also its utilization. The static or descriptive knowledge is implemented with frames, i.e. goals, operators, context, requests, and arguments. For heuristic criteria concerning choice, evaluation, initialization, and adjustment, production rules are used. *Ref IA to systems with both frames and production rules. Both knowledge representation schemes and mixed control (production systems and frame-based systems). Such an architecture is efficient for specialized tasks with predefined role for frames and production rules. Examples: CLASSIC for classification task... By opposition with general systems working with rules (OPS5), frames (KRL?). Ce qui concerne l'efficacité dans les 2 paragraphes suivants pourrait être repris dans la conclusion de la partie Implementation.* **Frames:** Frames are a very interesting way of structuring the knowledge base; grouping in the same location all the information about an algorithm or a method, is efficient (no inferences to execute), clear (readable), easy to access, and secure (a local modification will not have unwanted consequences as it happens when all the knowledge is expressed with rules). Various types of frames (goals, primitive or complex operators) structure the knowledge base into several levels of abstraction. So, incremental development of large knowledge bases by several specialists becomes possible. For instance, knowledge about chaining can be expressed by a specialist of this technique; then an expert on stereovision can build

a knowledge base which, if needed, refers to the goal *chaining* ; so, the goal *stereovision* will be partly solved using the knowledge base developed by the specialist in chaining.

Rule bases: The knowledge base is also structured using several small bases of production rules. Rules are grouped depending on their semantics (choice, evaluation, initialization and adjustment rules); in addition, these rules, which are typed, are attached to the only frames they are concerned with. Structuring the base in this way greatly improves the efficiency (only few rules are scanned at a given moment), and facilitates the development of knowledge bases by several experts (the same vocabulary can be used for different items: the same words *threshold* or *input image* can be used for the parameters of several distinct goals).

Figure 4 shows the **relations between the frames, and the rule bases.** A request (for instance *Request-00*) is related to a particular goal (here *Goal-1*); this goal can be solved by several operators (*Operator-1, Operator-2,* and *Operator-3* in this case). A decomposition of a complex operator (*Operator-2,* or *Operator-3*) is a tree of requests to other subgoals. Each goal, and each operator have two private rule bases: one rule base on choice, and one on evaluation for a goal; one rule base on initialization, and one on adjustment for an operator.

3.4.2 Implementation of the reasoning model

The control structures have five components : **a pilot**, **a planner**, **a parameter settler**, **a rule interpreter** (a general one called by the previous components), and **an interface to activate external programs.** The pilot has the role of a supervisor and of a controller: it controls the execution of a tree of requests, and decides to call other components for specific tasks. The algorithm presented in Figure 5 controls the execution of a tree of requests. Of course, to process the initial request, this algorithm works on a tree reduced to one request.

Step 1 (performed by **the planner**) is a global matching verifying that there exists at least one operator to solve the goals of the requests. Step 2 (performed by **the pilot**)

selects one request to process (when the tree is reduced to one request, this step is immediate). Step 3 (performed by **the planner**) eliminates invalid operators, and performs a classification of the valid ones using the choice **rules.** Step 4 (performed by **the pilot**) selects one operator among the valid ones. Before execution of a chosen operator (step 5), **the parameter settler** determines the values of its parameters: using initialization **rules** for the first attempt, using adjustment **rules** for the next ones. If the operator is a primitive one, the execution is done by **the interface with programs**; in the case of a complex one, this algorithm is recursively called to control the execution of the tree of requests of its decomposition. Then, according to the request, the evaluation of the results (step 6) is achieved automatically using the evaluation **rules,** or by graphic presentation of the results to the user who evaluates them interactively. In case of unsatisfactory results, another execution is performed after adjustment of the input parameters of the operator, or after choice of another operator.

3.4.3 Implementation: Conclusion

efficiency and bench marks ? Petits SE dans les objets

4 Examples of integration using OCAPI

In this section, we present two examples of expert systems built with OCAPI. The first example is the integration of a stereovisual process (including the extraction of the primitives); we show in detail how the various kinds of knowledge are expressed in the knowledge base. The second example is the description of a knowledge base developed for an application in astronomy; the integrated processing is a complex one.

4.1 Integration of a stereovisual process

This example has been developed within the framework of the Eureka project Prometheus. It deals with detection of obstacles in road scenes and urban scenes, using stereovision data. The data are taken from two cameras which are fixed on the top of a moving car. A pyramidal stereovision algorithm based on contour chain points [21] is used to reconstruct the 3D environment in front of the vehicle.

Integration of the knowledge on the use of this stereovisual process is needed for robustness, and above all to manage the great variety of the scenes. More precisely a lot of contextual values in the scenes vary, like luminosity (depending on the weather conditions and on the moment in the day), complexity of the scene (depending on the number of objects in front of the car), and velocity (depending on the location of the scene: highway, countryside road or urban street). Since the scenes may be quite various, it is not possible to fix the values of the different parameters involved in the processing.

The initial request is thus the stereovisual processing of a pair of images taken by two cameras. Figure 6 shows an example of such images in the case of an urban scene.

The pyramidal process works at several resolutions (to be precise, four resolutions); for each resolution, first an extraction of the primitives, then a matching of these primitives are performed; for each image (the left and the right one), extraction of primitives consists of contour detection, thresholding by hysteresis, computation of precise orientation, and contour chaining. Matching results obtained at a given resolution are interpolated, and used to reduce the search at the immediately higher resolution.

The current knowledge base is composed of 15 goals, at which are attached 19 operators. Among these operators, 14 are primitive ones (programs) and 5 are complex ones (with decomposition into steps). 50 production rules are attached to those frames: there are 6 choice rules, and 10 evaluation rules attached to the goals; 14 adjustment rules, and 20 initialization rules are attached to the operators. Thanks to 34 inferences rules for initialization and adjustment of parameters, this knowledge base can process fully automatically really different types of data. This knowledge base could be extended by

adding new alternative operators both for extraction of primitives and matching, with their corresponding rules.

4.2 PROGAL: an integration in astronomy

The problem of integrating image processing modules arises also for the development of specific applications. It is the case for applications which have to be flexible, i.e. the processing must be adaptable according to changes in the environment or to the type of data.

In this example, we are interested in automating the data processing of images containing a galaxy. The goal of this application is to class the galaxies according to a predefined classification. A synopsis of the global architecture of the system is shown in Figure 7. The first phase computes, from the image, numerical parameters describing the galaxy morphology. This phase is performed by PROGAL, an expert system using image processing programs; this system has been built with OCAPI, the expert system shell which has been previously presented. The second phase classes the galaxy described by the numerical parameters computed in the first phase, by using a taxinomy defined by the experts of the field. This phase provides as output the morphological type plus a detailed symbolic description of the galaxy; it is performed by the expert system SYGAL [22] built with CLASSIC [23], an expert system shell specialized in classification.

In this application the role of the image processing phase is to describe the morphology of the galaxy in terms of numerical parameters. For example, Figure 8 shows the input image *mt671n17.fl* containing in its center the galaxy *ngc4523*, and the numerical description of the galaxy computed by the processing. These parameters correspond, for instance, to measures of the orientation, the elongation, or the compactness of several regions of the galaxy. For each image, the image processing is approximately the same; it has five main stages: initialization (creation and initialization of the file containing the numerical parameters), extraction in the image of the object of interest (the galaxy), computing of global parameters describing the galaxy, building of iso-intensity contours

(corresponding to different regions of the galaxy), and for each of these contours, computing of numerical parameters describing them. Among these stages, some of them are relatively complex. For example, the isolation of the galaxy has three subparts: localization of the object, effective isolation, and noise removal. The object localization starts with a low resolution localization (by the research of an extended bright region and by correlation of this region with a photometrical model of a bulge [galaxy center]), and is refined at high resolution.

Because of the great variability in the images, which are taken by different observation instruments (photographic plates or CCD cameras, high or low resolution telescopes...) both the sequences of procedures, and the values of their parameters need to be adapted. Some examples of images are shown in Figure 9.

In this application, the set of procedures to pilot is composed of specific procedures (written in C and Fortran77), standard image processing procedures of a library (INRIMAGE), and operating system commands (Unix). Some procedures such as histogram computing, filterings (median, morphological), extraction of images, and convolutions are standard ones. Specialized algorithms are, for example, computation of some thresholds, isolation of the galaxy, or computation of morphological parameters.

4.2.1 The knowledge base

In this section, we present the knowledge base of PROGAL. The image processing procedures are detailed in [22]. The examples are expressed using the knowledge representation schemes of OCAPI: frames and production rules. As it has been shown, an OCAPI knowledge base is structured by frames belonging to different classes. Now, we detail some examples of the main frames in PROGAL.

Context frame : The frame *context* describes the possible values of the characteristics of the input data; this information is important because it is used in the rules (choice rules to decide pertinent operators in function of the context, initialization rules to adapt initial values of the parameters to the current context...). Presently, we use in

the production rules two kinds of criteria : the criteria related to the type of observation instrument as the level of noise in the input image, the nature of the image (density image provided by the sensor or intensity image obtained after a photometrical calibration) and the size of the objects which could partially occult the object of interest, and the criteria related to the semantic contents of the image, as the position of the object of interest in the image, the eventual presence of other objects (bright foreground stars) and their distribution. So, in this application, the context of the knowledge base takes into account these criteria, as it is shown in Table 1.

Goals frames: The knowledge base contains several goals such as: *morphological-description*, *object-extraction*, *global-param-computing*, *contour-building*, *contours-param-computing*, *coarse-object-extraction*, *coarse-object-detection*, *thresholding*, *contour-chaining*, *including-box*, *histogram*, *convolution*, *multiplication*....

For instance, the goal *coarse-object-extraction* [Figure 10] has one input data argument (the input image) and five output arguments (the x and y positions of the object, the subimage containing the object, and the size of the subimage in the x and y directions). Choice rules are attached to this goal (they are discussed in the next section); in this goal, there are no evaluation rule.

Operators frames: Several operators have been defined in the knowledge base: they correspond to a particular goal as shown in Table 2.

For instance, the operator *O-coarse-object-detection1* [Figure 11] which solves the goal *coarse-object-detection* has the same input argument as its goal (the input image), and the same output arguments (the x and y positions and sizes of the subimage containing the detected object as well as the size of the object). To detect the area containing the galaxy, this complex operator looks for a bright extended object; being a complex operator, it is decomposed into several substeps (requests to other goals): first thresholding, then filtering, then creation of the contour chain of the largest region, then computing of the position and size of the including box of this region. This operator is used in the object extraction phase in the case where there are no a priori information on the position of the

object in the image (the object is not centered in the image or its position is unknown). Initialization and adjustment rules determine the value of the input parameter *smuls* for each of the execution of this operator.

Request frames:

Each substep of the decomposition of the complex operator *O-coarse-object-detection1* is a request; Figure 12 shows the details of the first one which is a request to the goal *thresholding*. Control of data flow of input and output data of the operator, and of the subrequests is expressed directly in the requests and in the operator arguments.

We show in Figure 13 a request which specifies constraints on results; the constraints express that the size of the detected object must be greater than a threshold expressed in the context in order to discriminate it easily from eventual occulting objects.

Choice rules:

When several operators are available to solve the same goal, we have to express how to select the pertinent operator according to the context. For example, as we have seen in Table 2, two operators *O-centered-object-ext1* and *O-general-object-ext* can solve the goal *coarse-object-extraction*. These two operators perform the extraction of a subimage containing the object of interest; the first one, simply extracts a subimage in the center of the input image, subsuming that the object is located in the center of the image; the second operator in a first step looks for detecting the object before extracting the subimage (it is for this detection phase that we will show initialization, evaluation and adjustment rules). One choice rule attached to the goal *coarse-object-extraction* is shown Figure 14; this rule handles the particular case where there is an a priori knowledge on the location of the object. Other rules exist to adapt the choice of the operator to scenes for which there is no a priori information on the location of the object or if the object is not centered in the image.

Evaluation rules:

Most of the time, it is very difficult to express evaluation criteria to assess the results of low level goals (like *thresholding*) since no interpretation of the results is performed at

this stage; when a goal provides high level data as results, it is possible to find a criterion of evaluation based, for example on some measure. Two evaluation rules associated to the goal *coarse-object-detection* are shown in Figure 15; the first one states that, if the size of the detected object is too small w.r.t. the specifications expressed in the request when occulting objects are present, this detection is ambiguous and a failure is decided; this failure sets a loop mechanism which starts a new execution of the same operator with modified input parameters. On the right part of the same figure (Figure 15) another kind of evaluation rule is shown; this rule states that if the detection is not ambiguous but if the size of the object is very close to the ambiguity threshold called *min-specifs*, this detection is assessed as a limit case. This assessment will be used further in the reasoning but will not start an immediate failure.

Initialization rules:

Initialization of the parameters of the operators depends highly on the context. For instance, the operator *O-isolate* which performs the effective isolation of the galaxy by computing the exact boundary of the object of interest, needs the initialization of two input parameters: *sf1* and *sf2*. These parameters are two thresholds and their values depend on the nature of the image (i.e. density or intensity image); we show on Figure 16 one rule among those initializing these parameters w.r.t. the context. In the same way, the operator *O-coarse-object-detection-1* needs to initiate the value of the threshold *smuls*. This input argument is defined as being a parameter with numerical values between 0 and 1 and no default value. Its value depends on several other variables (*sm, saire1 and saire2*). So, initialization rules are necessary to define this value w.r.t. the context of use of the operator. One initialization rule of *smuls* is shown on Figure 17. If *sm* is lower than *saire2*, *smuls* is computed using an expression depending of *saire1* and *saire2*. These rules are triggered during the first execution of the operator, for a given request to the goal *coarse-object-detection*.

Adjustment rules:

When an operator fails because the results are not satisfactory, we need to express how

to modify its parameters w.r.t. the type of failure. There are three types of adjustment rules: the rules which select an adjustment method for a given parameter (*dichotomy method, percentage method...)*, the rules which determine the direction of the modification of the value for the next execution (*increase* or *decrease)*, and the rules which fix imperatively a new value. All these adjustment rules test the assessments which have been deduced by the evaluation rules which have concluded to the failure; so, based on these diagnoses the best adjustment for a given parameter is selected. The rule presented on the left side in Figure 18 selects an adjustment method for the parameter *smuls* of the operator *O-coarse-object-detection-1*; the method is a percentage technique with a coefficient of 5% and with *saire2* as minimal bound. The rule presented on the right side in Figure 18 expresses that when the detection is ambiguous, the threshold *smuls* has to be decreased. The way to decrease this parameter will be done according to the method previously selected.

4.2.2 Execution

In order to illustrate how the system works, we present an effective execution of a request on an image of the galaxy ngc4523 in the Virgo cluster (see Figure 9). We ask to the system to solve a request to the goal *morphological-description*: the input argument *mt671n17.fl* is the name of an image file containing the galaxy ngc4523. In this particular case, the location of the object in the image is unknown, the image is a density image, the image is noisy, there are stars anywhere in the image, and their size is lower than 5 pixels.

During the reasoning, the system builds dynamically the tree of the subgoals it solves (see Figure 19). The tree expresses the decomposition of the final processing into substeps up to primitive operators (programs). The level of abstraction is represented on the horizontal axis: from the most abstract on the left, to the least one (the concrete programs) on the right. The temporal ordering is represented on the vertical axis: the lowest leaf corresponds to the first step, the uppermost one to the last step. In this figure, nodes are goals to be solved. After an initialization phase, there is a phase of extraction of the

object which begins by an object detection phase. In order to perform this detection, the system computes some statistics on the density values, then it creates a low resolution image (input image: 512×512, sampling factor: 4), then it starts a coarse extraction of the object on this low resolution image. In order to perform a coarse extraction of the object, there is a phase of coarse detection of the object.

The results of the first execution of the operator *O-coarse-object-detection1* to solve the goal *coarse-object-detection* are shown in Figure 20. Note that the detected object is the top left star and not the central galaxy; in fact, the position of the detected object is given by the output arguments *ix* and *iy*, their values after this execution are respectively 32 and 16; the threshold image is a 128×128 image computed from the low resolution one. The size of the detected object (size=5) being too small, the rule shown in Figure 15 is triggered and a looping starts.

Once the adjustment rules attached to the operator *O-coarse-object-detection1* have been triggered, the parameter *smuls* has been decreased, and its value becomes 0.2592023 . The results of the second execution are displayed Figure 21. The size of the object is still too small. However, this time it is the galaxy which has been detected: the position of the detected object is given by the arguments *ix* and *iy,* and their values are respectively 64 and 62. It could be possible to stop the loop; but, without other evaluation criteria for this goal than the size of the object, the system loops again.

A third trial is performed with the value 0.2509509 for *smuls*. The results of the third execution are diplayed in Figure 22. It is still the galaxy which has been detected (position 62 and 61) and the size has increased. The parameter *smuls* having reached the minimal value, beyond which too many background noise is detected, it is impossible to perform a better object detection; so no more looping is required.

The reasoning continues as shown in Figure 19: the following phases of the coarse object extraction are the extraction of the subimage and the multiplication of this subimage with the binary thresholded image, then the bulge of galaxy is estimated by maximazing the correlation with a gaussian model and this estimation is refined based on this

estimation working at the highest resolution. The object extraction continues with the isolation of the object by computing its limits using a model of phometrical variation of the intensity from the bulge, and ends with a noise removal step. The other steps *global-param-computing, contours-building* and *contours-param-computing* are the computing of classical global shape parameters (*area, orientation, ellipticity)* and of two specific photometrical parameters (*profile and linear_err),* the computing of local parameters describing the shape and photmetry of the object more or less far from the center. For more details see [24,25].

During this execution 49 programs are run; among them 4 programs are executed 3 times each, during the evaluation-adjustment phases. For the execution of this request, choice rules and evaluation rules have used contextual information about the position of the galaxy in the image (centered or not), the presence of noise, the presence of stars, and the type of the images (density or intensity). As this information was not specified to the system in the initial request, the user has been asked for during the execution.

4.2.3 Conclusion

The current knowledge base is composed of 44 goals, at which are attached 47 operators. Among these operators, 28 are primitive ones (programs) and 19 are complex ones (decomposition into steps). 21 rules are attached to those frames. This example shows that it is possible to develop, with OCAPI, knowledge bases for specific applications, with a complex structure and numerous steps.

The interest of such a knowledge-based system has two main aspects: first, once a sequence of programs has been defined and tested on a set of data, it enables its utilization within a more general context; the expliciteness of the various kinds of knowledge which are usually implicit enhances the robustness of the method (versus other input data) or its generality (versus other applications); second, as the image processing expertise is included in the system, it enables the use of the software to user working at another level of abstraction (here an astronomer).

4.3 Building knowledge bases in integration

From the two examples previously described, we can deduce several points concerning the building of such knowledge bases in integration. In order to facilitate the building of a knowledge base, generally we must begin with the description of the static knowledge: first, this knowledge is the most stable; second, this knowledge is the support for dynamic knowledge. In the case of integration, static knowledge consists in the context, the goals, the operators, and the requests, and dynamic knowledge consists in the choice, evaluation, initialization, and adjustment rules.

Static knowledge: Developing PROMETHEE and PROGAL presented in the previous sections, has shown the necessity of beginning with the description of all available programs, which, in fact, are the basic building blocks of the knowledge. So frames describing the primitive operators are the first to be expressed. Next step is developing of higher levels of abstraction operators; depending on applications, particular sequences of processing can be expressed.

Dynamic knowledge: Developing of the dynamic knowledge is a more complex process; it may imply developing new algorithms for, for example, initialization of parameters or evaluation of results. It may also imply the extension of the knowledge base or modifications of some programs because, at this step, the expert usually will be conscious that a great deal of knowledge is implicit inside the existing programs; so, he or she will try to make them explicit in the knowledge base.

5 Discussion

We can notice some emerging concepts from the various works on integration of image processing procedures. Four main topics are relevant: knowledge expression, planning facilities, execution capabilities, and control of execution functionnalities. Table 3 summarizes the main concepts in each topic for each of the systems presented in this article. The symbol $\star$ denotes the presence of the concept in an explicit way in the system, while

the symbol $-$ denotes its absence. The symbol $\star\star$ means that the system offers really good functionnalities concerning the concept, usually offering various means to perform the associated task.

As shown on the section 2, all these systems have the architecture of an expert system; in fact, they offer quite good functionnalities to express the knowledge.

The planning facilities of the systems are based on the notion of hierarchical squeletton, which can be raffined or extended. Two systems (Expert assistant and Toriu) don't offer replanning facilities, while DIA-ES has really good capabilities.

The effective execution of the image processing procedures is performed by five of the systems ; each of them offer mechanisms to initialize the parameters.

Expert assistant and Toriu's system have no facilities for the control of execution. In the other systems, the evaluation and the adjustment of the parameters have usually to be performed by the user. In addition, in OCAPI, when available, knowledge of evaluation of results, and of adjustment of parameters is expressed using rules; so evaluation and adjustment can be automatic.

6 Conclusion

We have introduced the notion of semantic integration of programs. Then, a model of reasoning and knowledge involved in image processing has been proposed. An implementation of this model, OCAPI, which has the architecture of an expert system generator, has been described.

Two examples of utilization of OCAPI have been shown: a knowledge base on a stereovisual process, and another one for morphological description of galaxies. Currently, several other knowledge bases are under development on various domains, such as on stellar astrophysics (planning and control of execution of numerical algorithms), and on remote sensing (an industrial application).

Various other tools performing semantic integration of programs have been discussed. There is not yet a system including all these concepts. More particularly, OCAPI, which includes most of them does not yet have facilities for dynamic modification of the skeleton, as DIA-ES has. For instance, to perform such modification could be useful for facultative operations as filtering for noise removal. On the other hand, OCAPI offers the most developed trial-and-error mechanisms (evaluation of results, and adjustment of parameters).

But : automatiser les traitements, faciliter l'expresison de la connaissance (expertise). Mais pas construction automatique de nouveaux traiteemnts et/ou génération de code. POur cela, il faut étdnre les facilités de planification. Par xemple :

- caractéristiques des opérateurs (choix)
- construction de décomposition / modifications

CONTEXT ITEM	POSSIBLE VALUES	COMMENTS
noise	*low important*	noisy image or not
image-type	*intensity density*	image type
min-size	$[1, 1000000]$	minimal size of the object expressed in pixels (fonction of the observation instrumentation)
object-position	*centered uncentered*	position of the object in the image
other-objects	*in-front-of-the-galaxy absent everywhere outside-the-galaxy*	eventual presence of other objects such as stars

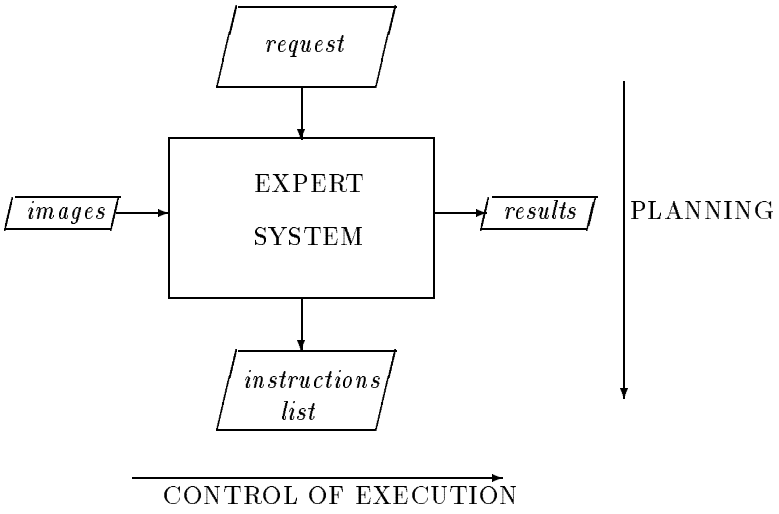
- Table 1 - Definition of the context in the PROGAL base

goal	**operators**
coarse-object-extraction	O-centered-object-ext1, O-general-object-ext
coarse-object-detection	O-coarse-object-detection1
object-isolation	O-isolate
filtering	O-morpho-f, O-median-f
...	...

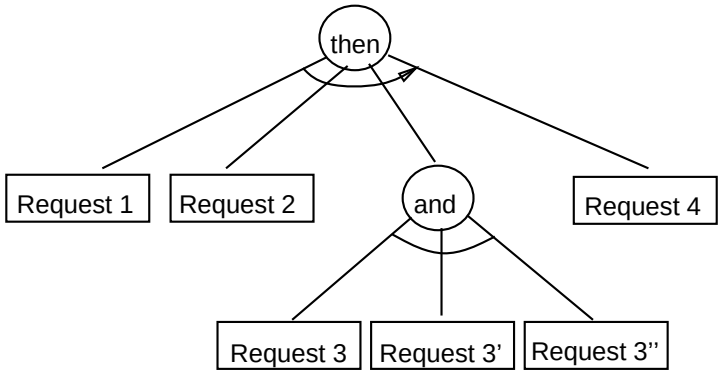
- Table 2 - Some goals and their associated operators from the PROGAL base

Systems \ **Concepts** \	Expert assistant	Toriu	EXPLAIN	DIA-ES	Bailey	OCAPI
Knowledge Expression						
explicit description of operators	⋆	⋆	–	⋆	⋆⋆	⋆⋆
levels of abstraction	⋆	⋆⋆	⋆⋆	⋆⋆	⋆	⋆⋆
description of data and context	⋆	⋆	⋆	⋆	–	⋆
Planning Facilities						
planning	⋆	⋆	⋆	⋆⋆	⋆⋆	⋆⋆
replanning	–	–	⋆	⋆⋆	⋆	⋆
Execution						
effective execution	–	⋆	⋆	⋆	⋆	⋆
parameters initialization	–	–	⋆	⋆	⋆⋆	⋆⋆
Control of Execution						
results evaluation	–	–	⋆	⋆	⋆	⋆⋆
parameters adjustment	–	–	⋆	⋆	–	⋆⋆

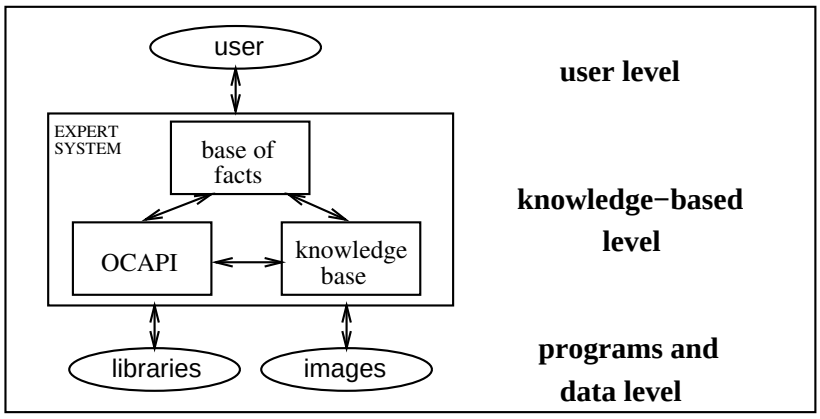
- Table 3 - Concepts and systems in ...


request
EXPERT
SYSTEM
images
results
PLANNING
instructions
list
CONTROL OF EXECUTION

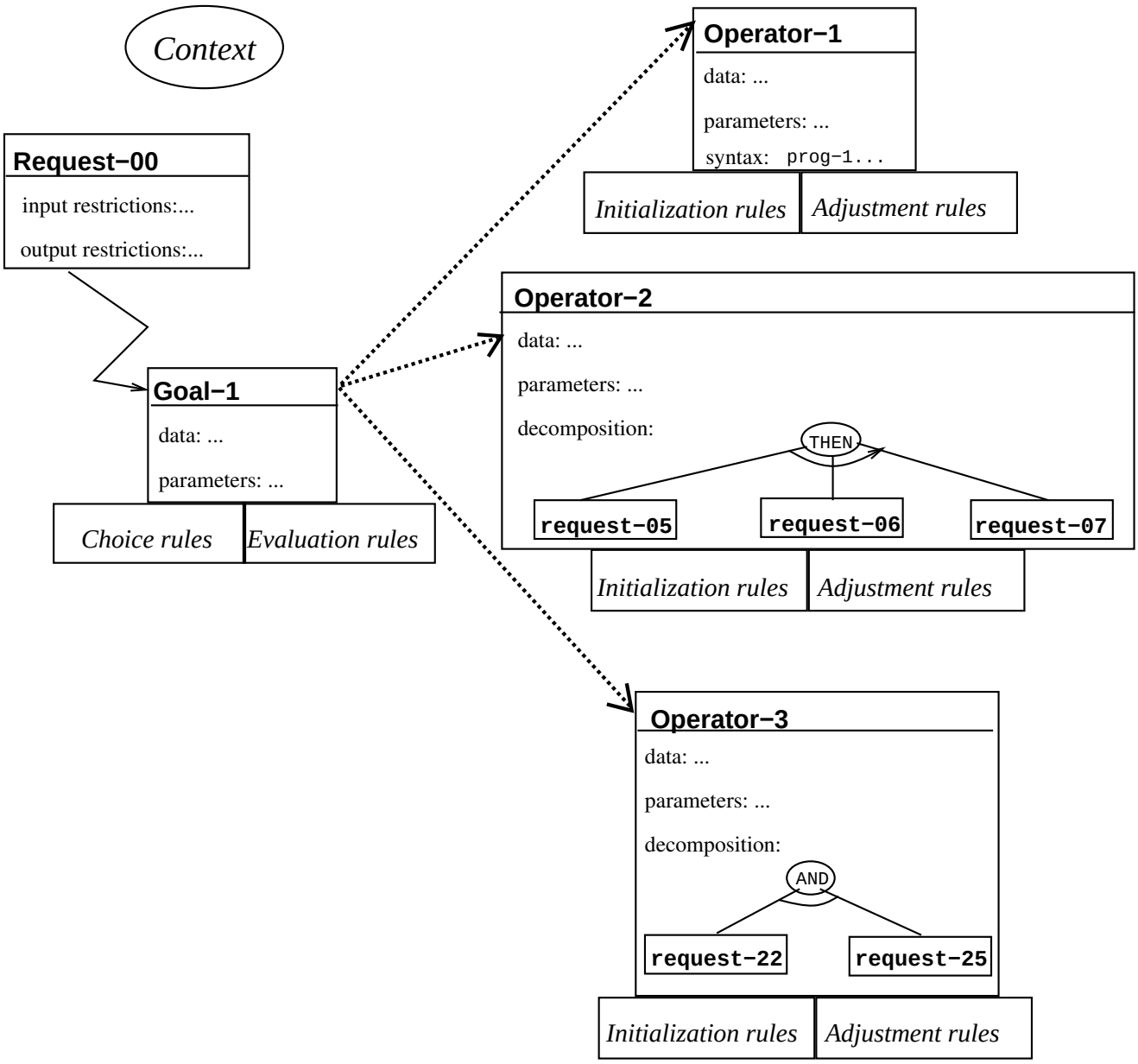
- Figure 1 -


then
Request 1
Request 2
and
Request 4
Request 3
Request 3'
Request 3''

- Figure 2 -


user
EXPERT
SYSTEM
base of
facts
OCAPI
knowledge
base
libraries
images
user level
knowledge−based
level
programs and
data level

- Figure 3 -


Context
Request−00
input restrictions:...
output restrictions:...
Goal−1
data: ...
parameters: ...
Choice rules
Evaluation rules
Operator−1
data: ...
parameters: ...
syntax: prog−1...
Initialization rules
Adjustment rules
Operator−2
data: ...
parameters: ...
decomposition:
THEN
request−05
request−06
request−07
Initialization rules
Adjustment rules
Operator−3
data: ...
parameters: ...
decomposition:
AND
request−22
request−25
Initialization rules
Adjustment rules

- Figure 4 -

-1- global matching

while not all requests have been processed

-2- selection of the request to be processed

-3- classification of the operators (using choice rules)

while the request is not satisfied

-4- selection of the best operator

-5- execution of the operator (using initialization or adjustment rules)

-6- evaluation of the results (using evaluation rules)

- Figure 5 -

- Figure 6 -

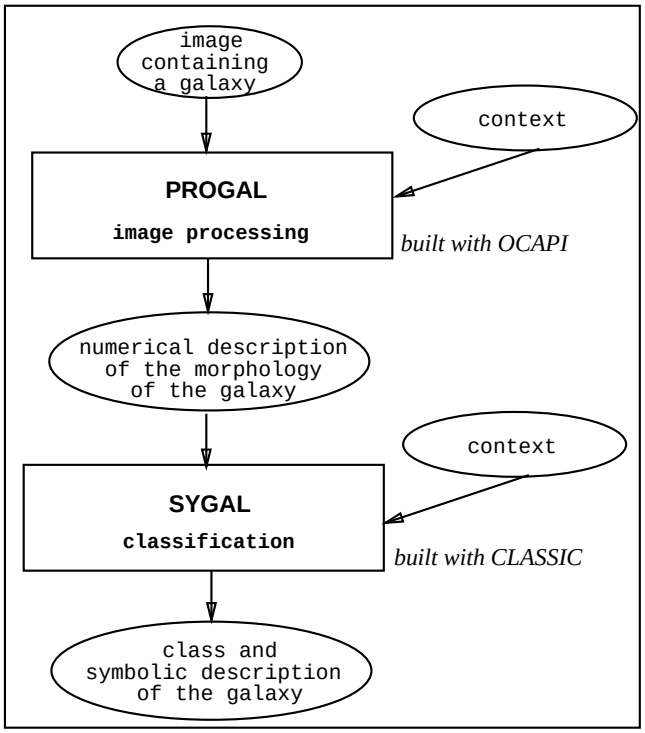


image containing a galaxy
context
PROGAL
image processing
built with OCAPI
numerical description of the morphology of the galaxy
context
SYGAL
classification
built with CLASSIC
class and symbolic description of the galaxy

- Figure 7 -

```
galaxy :  ngc4523 :
          area              : 4536.5 ;
          ellipticity       : 0.76 ;
          linear_err        : 0.469 ;
          profile           : 1.08 ;
          orientation       : 53.74 ;
          contour c1 : {    centre_err      : 2 ;
                            ellipse_err     : 0.01 ;
                            compactness     : 1. ;
                            angle           : -3.4 ;
                            eccentricity    : 0.62 };
          contour c2 : {    centre_err      : 2 ;
                            ellipse_err     : 0.16 ;
                            compactness     : 1.4 ;
                            angle           : -10.1 ;
                            eccentricity    : 0.26 };
          contour c3 : {    centre_err      : 2 ;
                            ellipse_err     : 0.11 ;
                            compactness     : 2.4 ;
                            angle           : -25.1 ;
                            eccentricity    : -0.01 };
          contour c4 : {    centre_err      : 34 ;
                            ellipse_err     : 0.47 ;
                            compactness     : 13.8 ;
                            angle           : 47.1 ;
                            eccentricity    : 0.27 };
          contour c5 : {    centre_err      : 19 ;
                            ellipse_err     : 0.64 ;
                            compactness     : 15.1 ;
                            angle           : -88.1 ;
                            eccentricity    : 0.4 }.
```

- Figure 8 -

ngc4523

ngc4473

ngc7531

ngc6946

- Figure 9 -

Goal	**coarse-object-extraction**
input data :	image
output data :	x-pos, y-pos, subimage, x-size, y-size
choice rules :	{ R1, R2 }
evaluation rules :	{ }

- Figure 10 -

Operator	**O-coarse-object-detection1**
goal	**coarse-object-detection**
input data :	image *"initial image"*
parameters :	smuls
output data :	ix *"x position"*
	iy *"y position"*
	x *"x size"*
	y *"y size"*
	size *"object size"*
decomposition :	request r-thresh
	then request r-filt
	then request r-contour-chain
	then request r-including-box
initialization rules :	{ RI-dzg-1, RI-dzg-2 }
adjustment rules :	{ RA-dzg-3, RA-dzg-4 }

- Figure 11 -

Request	r-thresh	**goal :**	thresholding	
constraints :	**input :**	ie	=	image OF OPERATOR O-coarse-object-detection1
		thresh	=	smuls OF OPERATOR O-coarse-object-detection1

- Figure 12 -

Request	r-cod	**goal**	coarse-object-detection
constraints :	**input :**	image	= image OF OPERATOR O-general-object-ext
	output :	size	> min-size

- Figure 13 -

IF *object-position centered*
THEN *:use-operator O-centered-object-ext1*

- Figure 14 -

IF	*:assessed size too-small/specifs*	IF	*:assessed detection not ambiguous*
and	*no other-objects absent*	and	*size < min-specifs + 2*
THEN	*:assess detection ambiguous*	THEN	*:assess detection limit*
and	*:failure*		

- Figure 15 -

IF	*image-type density*
THEN	*:initialize 'sf1 0.1*
and	*:initialize 'sf2 1*

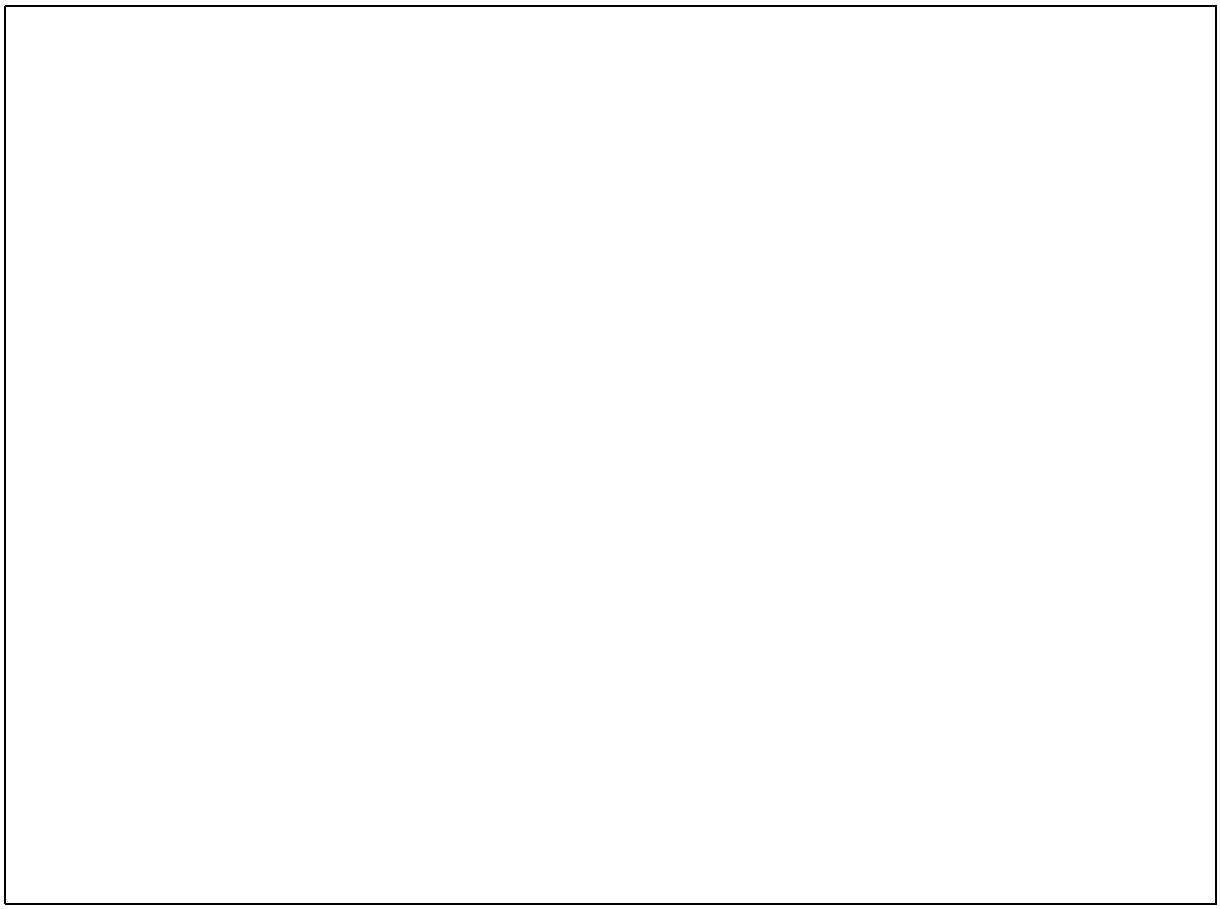
- Figure 16 -

IF $sm < saire2$
THEN $:initialize\ 'smuls\ (saire1 + (saire2 - saire1)/3)$

- Figure 17 -

IF	...	IF	*:assessed detection ambiguous*
THEN	*:adjust-using '% 'smuls*	THEN	*:decrease smuls*
and	*:%-coeff 'smuls 0.05*		
and	*:%-min 'smuls 'saire2*		

- Figure 18 -



Left to right: from the initial request to concrete programs.

Bottom to top: from the first substep to the last substep.

- Figure 19 -

detected object

thresholded image

- Figure 20 -

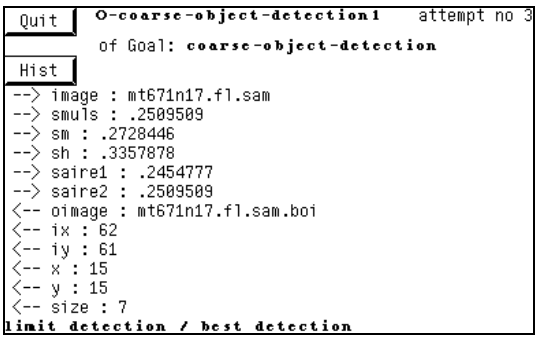
detected object

thresholded image

- Figure 21 -

detected object

thresholded image



- Figure 22 -

Figure captions

Figure 21: Second execution of the goal *coarse-object-detection*

Figure 22: Third execution of the goal *coarse-object-detection*

Footnotes

[1] Le_Lisp is a registered trademark of INRIA

References

[1] A.M.Nazif and M.D.Levine, Low Level Image Segmentation: An Expert System, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6.5**, 1984, 555-577.

[2] T.Matsuyama, Expert systems for image processing: Knowledged-based composition of image analysis processes, *Comput. Vision Graphics Image Process.* **48**, 1989, 22-49.

[3] A.Hanson and E.Riseman, The VISIONS image-understanding system, in *Advances in Computer Vision,* (C.M.Brown, Ed.), pp.1-114, Erlbaum Assoc, 1987.

[4] R.A.Brooks, Symbolic Reasoning Among 3-D Models and 2-D Images, *Artificial Intelligence Journal* **17**, 1981, 285-348.

[5] M.Thonnat and M.H.Gandelin, An expert system for the automatic classification and description of zooplanktons from monocular images, in *Proceedings, 9th Int. Conf. on Pattern Recognition, Roma*, 1988, pp.114-118.

[6] K.Ikeuchi and T.Kanade, Automatic Generation of Object Recognition Programs, in *Proceedings of the IEEE* **78.8**, 1988, pp.1016-1035.

[7] M.D.Johnston, An expert system approach to astronomical data analysis, in *Proceedings, Goddard Conf. on Space Applications of Artificial Intelligence and Robotics,* 1987, pp.1-17.

[8] T.Toriu, H.Iwase and M.Yoshida, An Expert System for Image Processing, *FUJITSU Sci.Tech.J.* **23.2**, 1987, 111-118.

[9] T.Tanaka and N.Sueda, Knowledge acquisition in image processing expert system EXPLAIN, in *Proceedings, Int. Workshop on Artificial Intelligence for Industrial Applications*, Hitachi City, 1988, pp.267-272.

[10] H.Sato, Y.Kitamura and H.Tamura, A knowledge-based approach to vision algorithm design for industrial parts feeder, in *Proceedings, IAPR Workshop on Computer Vision - Special hardware and industrial applications,* Tokyo, 1988, pp.413-416.

[11] D.G.Bailey, Research on computer-assisted generation of image processing algorithms, in *Proceedings, IAPR Workshop on Computer Vision - Special Hardware and Industrial Applications,* Tokyo, 1988, pp.294-297.

[12] V.Clément and M.Thonnat, Handling knowledge on image processing libraries to build automatic systems, in *Proceedings, Int. Workshop on Industrial Applications of Machine Intelligence and Vision*, Tokyo, 1989, pp.187-192.

[13] Intellicorp, *KEE System Manual,* Menlo Park, CA, 1985.

[14] H.Tamura and K.Sakaue, DIA (Digital Image Analysis) - Expert System : an approach to future vision system design, in *Int. Symp. on Image Processing and its Applications,* Tokyo, 1984.

[15] K.Sakaue and H.Tamura, Automatic generation of image processing programs by knowledge-based verification, in *Proceedings, IEEE on Computer Vision and Pattern Recognition,* San Francisco, 1985, pp.189-192.

[16] H.Tamura *et al.*, Design and implementation of SPIDER - a transportable image processing software package, *Comput. Vision Graphics Image Process.* **23**, 1983, 273-294.

[17] D.G.Bailey and R.M.Hodgson, VIPS - a digital image processing algorithm development environment, *Image and Vision Computing* **6.3**, 176-184, Butterworth and Co. (Publishers) Ltd, 1988.

[18] R.C.Vogt, Formalized Approaches to Image Algorithm Development Using Mathematical Morphology, in *Proceedings, VISION'86,* Detroit, 1986.

[19] J.Hendler, A.Tate and M.Drummond, AI Planning: Systems and Techniques, *AI Magazine* **Summer 1990**, pp.61-77.

[20] A.Barr, P.R.Cohen and E.A.Feigenbaum, *Handbook of Artificial Intelligence,* Pitman, 1982.

[21] A. Meygret, M. Thonnat and M. Berthod, A pyramidal stereovision algorithm based on contour chain points, in *Computer Vision - ECCV 90,* Lecture Notes in Computer Science **427**, (O.D.Faugeras, Ed.), pp.83-88, Springer-Verlag, 1990.

[22] M.Thonnat and A.Bijaoui, Knowledge-Based Classification of Galaxies, in *Knowledge-Based Systems in Astronomy,* Lecture Notes in Physics **329**, (F.Murtagh and A.Heck Eds), pp.121-159, Springer-Verlag, 1989.

[23] ILOG, CLASSIC Manuel de l'utilisateur, Paris, 1987.

[24] M.Thonnat, *Automatic Morphological Description of Galaxies and Classification by an Expert System*, Research Report INRIA No 387, March 1985.

[25] M.Thonnat, Toward an Automatic Classification of Galaxy, in *The world of galaxies,* Springer Verlag, pp. 53-74, 1989.