Evaluating PSMs in Evolutionary Design:

The AUTOGNOSTIC Experiments

Eleni Stroulia\textsuperscript{1} and Ashok K. Goel\textsuperscript{2}

\textsuperscript{1} Department of Computing Science,
615 General Services Building, University of Alberta, Edmonton, AB, CANADA T6G 2H1
stroulia@cs.ualberta.ca.

\textsuperscript{2} College of Computing,
Georgia Institute of Technology, Atlanta, GA 30332-0280, USA
goel@cc.gatech.edu

Abstract

The specification of generic Problem-Solving Methods has been a fertile research area. A lot of work has been devoted to developing languages for describing PSMs, identifying PSMs, and using their specifications for requirements capture, design and development of knowledge-based systems. In our work, we have been investigating another potential use for PSMs, namely, supporting the redesign of systems that fail to exhibit the behaviors desired of them, that is, behaviors similar to, but slightly different from, the ones they were originally designed to exhibit. To this end, we have defined a PSM modeling language and a failure-driven redesign process based on this language, both of which were implemented in the AUTOGNOSTIC system. In this paper, we report on a sequence of experiments performed with AUTOGNOSTIC. Some of them were exploratory and their goal was to enable the precise characterization of issues relating to the problem of system redesign, while others were designed to evaluate the PSM language and the redesign process implemented in AUTOGNOSTIC.
1 Motivation and Background

Research on Problem-Solving Methods (PSMs) focuses on abstract specifications of prototypical problems and the reasoning methods for solving them. A lot of attention has been given to the issues of developing languages for specifying PSMs and using PSMs specifications for supporting the development of knowledge-based systems (Clancey 1985; Chandrasekaran 1983; Chandrasekaran 1989; Steels 1990; Wielinga & Breuker 1986; Wielinga et al. 1992; Musen et al. 1993; Brazier et al. 1994; Fensel & Schoenegge 1997), but the issues of system maintenance and redesign have been paid much less attention comparatively. However, these later phases of software life-cycle can potentially be very lengthy and costly, thus making the problem of supporting them extremely important. Our work in the AUTOGNOSTIC project has focused on the use of PSMs for supporting the task of redesigning a system in these later phases of its life-cycle.

The need to redesign an existing system may arise as a result of an error discovered during a verification or a validation study. In the former case, the error signifies divergence between the actual and the intended behavior of the system, and the system’s implementation has to be modified to meet its design specification. In the latter case, the error signifies a discrepancy between the actual and the desired behavior of the system. It is possible that the system design may explicitly prescribe or proscribe the desired behavior under question, or it may completely ignore it. In the first case, the error is both a verification and a validation error, and its treatment is similar to regular verification errors, i.e., its implementation must be modified to enable the desired behavior to occur. If the system design explicitly proscribes or ignores a desired behavior, whose violation is revealed during validation, then both the system design specification and its implementation have to adapted to eliminate the discrepancy in question.

The problem of redesign due to validation failures is pervasive. A system may need to be redesigned even after it has been verified and validated once. The users’ behavioral requirements on any system deployed in a dynamic, real-world environment are bound to evolve. Or alternatively, a system may be re-deployed in a new environment, in which it may face requirements slightly dif-
ferent from the ones imposed on its original development. In this paper, we focus on the problem of system redesign due to a particular class of validation failures, that is, due to the system’s failure to deliver desired variations on its original behaviors. The high-level hypothesis of this work is the following:

**Hypothesis 0:** The specification of a system’s design in terms of its PSMs can support its redesign when it fails to deliver the behavior desired of it.

To evaluate this hypothesis, we had to

1. define a language for describing the design of a system in terms of its PSMs,

2. design a process for system redesign based on a system description in this language, and

3. refine this process by precisely characterizing the types of failures that it should be able to address, the types of adaptations that it should be able to perform, and the affects that it should have on the system’s behavior.

To this end, we developed AUTOGNOSTIC, a “shell” system embodying the SBF-TMK (Structure-Behavior-Function models of Tasks-Methods-Knowledge) language for specifying a system’s PSMs and a redesign process based on it. We then performed an extensive sequence of exploratory experiments with AUTOGNOSTIC redesigning ROUTER, a path-planning system, and KRITIK2, a design system, to sufficiently refine this high-level hypothesis so that it could be tested.

At the end of this exploratory research phase, we designed two experiments for evaluating the effectiveness of AUTOGNOSTIC’s process in redesigning a system to deliver solutions of improved quality. The positive results of these experiments led us to formulating more questions on the generality and the realism of this redesign process. These questions in turn led to further exploratory experiments with new test-bed systems. In this paper we report on all these experiments with AUTOGNOSTIC, their rationale, their design, and their results. For more detailed descriptions of the particulars of each experiment, the interested reader should refer to (Stroulia 1995).

The overall purpose of the first exploratory phase was to guide the evolution of the SBF-TMK lan-
guage and the redesign process into a mature and stable design. The systems chosen as test-beds for this phase, ROUTER and KRITIK2, were representative of the class of systems to which AUTOGNOSTIC’s modeling language and redesign process should apply. They were both developed under the task-structure methodology and well-designed feedback could be provided by the environment to characterize their desired behavior. The purpose of the second exploratory phase was to challenge the SBF-TMK language and the redesign process and to investigate the limits of their expressiveness and applicability. Therefore the systems chosen as test-beds in this phase were less “benign” than the first set of test-beds: they were both designed under different methodologies, their behaviors had to conform with quite strict real-time requirements, and the environment could provide no feedback with respect to what precisely their desired behavior should be. In a sense the two different sets of test-beds were used as a cross-validation mechanism for the overall research on AUTOGNOSTIC.

The rest of the paper is organized as follows: sections 2 and 3 discuss the language of SBF-TMK models and the failure-driven redesign process based on this language. Section 4 discusses our initial exploratory research in the context of ROUTER and KRITIK2, the first two test case systems that have been integrated with and benefited from AUTOGNOSTIC. Section 5 explains the latter experiments with AUTOGNOSTIC’s integration with ROUTER, which were designed to precisely evaluate the effectiveness of AUTOGNOSTIC’s redesign process. Section 6 describes the subsequent integration of AUTOGNOSTIC with two robotics systems, REFLECS and DAVID, and the experiments performed with them. Section 7 reviews the literature relevant to this work and puts its results in perspective. Finally, Section 8 summarizes our experiments and draws some conclusions regarding evaluation in PSMs research, in general.

2 SBF-TMK Models

SBF-TMK models are hierarchical state-transition models. Originally based on a language for describing how physical devices work (Goel 1989), they represent a system’s internal processing
as information flow through the system’s task structure (Chandrasekaran 1989). A task (first row of Table 1) is characterized in terms of the types of information (third row of Table 1) it consumes as input, the types of information it produces as output, and its functional semantics i.e., a partial specification of the nature of the transformation it is intended to accomplish between the two. An elaboration of the nature and the role of a task’s functional semantics is required here. First, it should be noted that the term “functional semantics” refers to the invariants of the information transformation that the task performs between its input and output, and should not be confused with the same term in programming languages theory. Furthermore, it is important to explain the notion of “partial semantics”: the task semantics is a specification of the behavior intended of the task by the designer. A design specification may be implemented in several alternative ways, each of which introduces additional low-level commitments that may impact the task behavior but should not contradict the task semantics. The task semantics is a partial specification, in that it does not imply a unique task implementation but rather a class of implementations, all of which conform with the intention of the system designer about the role that the task in question should play in the system.

A task can be accomplished by one or more methods, each of which decomposes it into a set of simpler subtasks. The system’s operators, i.e., its primitive design elements, constitute the elementary building blocks for all the tasks that it is capable of accomplishing. In addition to its functional semantics, the information-transformation function of a task can be characterized by specifying this task as an instance of another prototype task. Then this task inherits the prototype’s functional semantics, and can be accomplished by all methods applicable to the prototype.

A method (second row of Table 1) is characterized by the conditions of its applicability, the subtasks in which it decomposes the task it is applied to, and the control it exercises over their processing. A method is essentially a detailed information flow network, explaining a complex task’s information transformation. The state transitions in this network are carried out by the method’s subtasks. The whole network is indexed by the task to which the method is applicable. Thus, a system’s internal problem-solving mechanism is specified by its task structure, i.e., the recursive
decomposition of its overall task in terms of methods and subtasks into a set of elementary, *leaf*, tasks, directly accomplished by the system’s domain operators.

In an extension of the task-structure framework, SBF-TMK models also capture a meta-model of the system’s domain, in terms of domain concepts, and relations and higher-order constraints applicable among them. The system’s known domain concepts are organized in a hierarchical taxonomy. A concept (fourth row in Table 1) is characterized by its attributes, a predicate defining identity among its instances, and the range of its possible values. A domain relation (fifth row in Table 1) is specified by the concepts it applies to, a pointer to its evaluation mechanism (either a truth table or a predicate), and potentially a predicate evaluating its inverse relation. Finally, a domain constraint (sixth row in Table 1) is specified by pointers to the relations to which it refers and the relation it imposes between them.

[Table 1 about here.]

Together, the system’s task structure (in terms of tasks and methods) and its domain meta-model (in terms of concepts, relations, and constraints) constitute its SBF-TMK model.

3 Defining Failure-Driven Adaptation:

**The Exploratory AUTOGNOSTIC Experiments**

Because our goal was a system-independent redesign process, we developed AUTOGNOSTIC as a shell, which, when integrated with a system, can monitor its behavior and appropriately adapt it when it fails. To integrate a system with AUTOGNOSTIC, means to provide it with the SBF-TMK model of the system and to “link” a library consisting of the components implementing the system’s leaf tasks to AUTOGNOSTIC.

In the integrated system, AUTOGNOSTIC operates in two distinct modes: (a) monitoring the system behavior and (b) reasoning about it, experimenting with the system and redesigning it. In
monitoring mode, AUTOGNOSTIC controls the focus of the computation and the information flow among the system’s tasks according to the system’s SBF-TMK model. In a sense, AUTOGNOSTIC acts as an interpreter of the SBF-TMK specification of the object system. At the same time, it records the actual flow of control and information among the system tasks. The reader should not form the impression here that the object system could not exist and operate in the absence of AUTOGNOSTIC. The main objective of the monitoring process is to record the information and control flow among the system tasks, which could be accomplished by instrumenting the system with “sensors” recording the tasks order of execution and their input and output information. Our particular choice of a tightly coupled integration mechanism was motivated by two factors: one is its appropriateness for conducting experiments on run-time system reconfiguration and the other is the simplicity and ease of its implementation. In fact, in section 6.5 we discuss an alternative integration mechanism, more loosely coupled, and we describe AUTOGNOSTIC’s process in this scenario. In redesign mode, AUTOGNOSTIC reasons about the trace of the system’s behavior, as recorded during monitoring, in order to identify the potential causes of its failure and to propose redesign operations which could eliminate this failure.

3.1 Task Monitoring and Failure Detection

When a system is integrated with AUTOGNOSTIC, in the tightly coupled sense, problem solving is accomplished by traversing the system’s task structure as captured in its SBF-TMK model. When AUTOGNOSTIC visits a non-leaf task, starting from the root of the task structure, it evaluates the applicability conditions of its methods in the current state of the system’s environment. If multiple methods are applicable, AUTOGNOSTIC selects one at random and keeps the rest of them as alternatives in case the selected one fails. Then, it updates its task agenda with the subtasks of the selected method. If a task is specified as an instantiation of some prototype(s), AUTOGNOSTIC evaluates the conditions under which these prototypes can be accomplished and selects one whose applicability conditions are met, while remembering the others as alternatives. Finally, when AU-
TOGNOSTIC reaches a leaf task, it proceeds to invoke the operator that accomplishes it. Once a task is accomplished, AUTOGNOSTIC evaluates whether the information it produced matches its functional semantics and then proceeds to accomplish the next one on in its agenda, until this agenda becomes empty. Thus, AUTOGNOSTIC controls and monitors the information and control flow of the system’s problem solving.

3.2 Task-structure Redesign: Blame Assignment, Modification, Verification

AUTOGNOSTIC is able to deal with three types of failures:

1. violation of a task semantics.

2. problematic patterns of behavior indicating lack of progress, and

3. failures to produce a solution desired by the system’s users.

AUTOGNOSTIC recognizes semantics violations by evaluating whether the information produced by a task conforms with its semantics. Its response to this type of failure is not covered in this paper since it is essentially a verification failure. The interested reader could find a detailed treatment of this issue in (Stroulia 1995). The latter two types of failure are both evidence of undesirable behavior. AUTOGNOSTIC’s response to the second failure type is discussed in detail in Section 6.4. Finally, AUTOGNOSTIC recognizes a failure of the third type when it receives feedback from the environment suggesting an alternative solution, preferable to the one produced by the actual problem solving. In these cases, it proceeds to identify why this solution was not produced.

The blame-assignment process (Stroulia & Goel 1996) is a successively focused examination of the tasks involved in the production of the undesired behavior. Beginning with the system’s overall task, AUTOGNOSTIC evaluates whether the feedback and the task’s actual input validate the task’s functional semantics. If this is the case, it infers that the task in question should have produced the feedback, and the reason why it did not must lie within the subtasks in which it was
decomposed. Based on the problem-solving trace, the last such subtask is identified and the blame assignment focuses on it. If the functional semantics of a particular task is violated by the feedback and its actual input, the blame-assignment method attempts to infer alternative inputs which would make the feedback value valid with respect to the task’s semantics. If such alternative inputs can be found, the actual inputs are considered undesirable, and the blame-assignment focuses on the earlier subtask responsible for their production.

AUTOGNOSTIC’s blame-assignment process is able to identify errors in the specification of the system’s operators, in the organization of the task structure, and in the system’s domain knowledge. Once a (set of) potential cause(s) has been identified, AUTOGNOSTIC decides to address the most grave one. To this end, it may re-specify its domain operators (Strouila & Goel 1997), or it may modify the control among the system’s tasks, or it may modify the system’s domain knowledge.

Respecification, i.e., redesign, of the system operators occurs when the blame-assignment process has reached an erroneous operator. In this context, an erroneous operator is a leaf task whose semantics is either violated by the feedback solution and its actual input and no alternative input can be found to make the feedback valid, or it is verified by both its actual input and actual output as well as its actual input and the feedback. In the former case, AUTOGNOSTIC postulates that a potential cause for the failure might be the over-constrained semantics of the task that prevent it from delivering the feedback. The feedback solution exemplifies a set of quality requirements which, although in accordance with the overall behavior expected by system, conflict with the specification of the behavior delivered by a low-level operator. Thus, the feedback reveals an error in the information transformation that this particular operator was designed to accomplish in the context of the overall task, since it prevents the production of an acceptable and desired solution.

In the latter case, AUTOGNOSTIC postulates that the cause might be the under-specified semantics of the task which allows both the actual and the desired output to be produced, when only the latter conforms with the requirements on the solution quality, as exemplified by the feedback. Control modifications occur when AUTOGNOSTIC discovers through experimentation that, for the accomplishment of a particular complex task, an alternative method might have led to the desired
feedback. Domain-knowledge modifications occur when erroneous operators could have produced the desired behavior, had the domain knowledge on which they operate been different.

Once a modification is chosen and completed, AUTOGNOSTIC proceeds to verify the usefulness of the modification by executing and monitoring the failed mission once again. If this time the mission is successfully completed, then the modification is considered correct. Otherwise, AUTOGNOSTIC starts a new cycle of blame assignment, modification and verification until an effective modification is completed, or no other potential cause can be identified.

These steps of AUTOGNOSTIC’s redesign process were successively defined and fine-tuned during the initial exploratory experiments phase of AUTOGNOSTIC’s integration with ROUTER and KRITIK2 described in the next section.

4 Initial Exploratory Experiments

The initial formulation of our research hypothesis was quite abstract. It was, therefore, necessary to refine it so that it could be evaluated. To this end, the following issues had to be investigated:

**Issue 0.1:** Is the SBF-TMK language sufficiently expressive to describe “interestingly” complex systems? What is the effort required in modeling a system in SBF-TMK?

**Issue 0.2:** What class of redesign problems should AUTOGNOSTIC’s problems be able to address? What types of adaptations should it be capable of?

**Issue 0.3:** What precisely are the affects of AUTOGNOSTIC’s redesign process to the system’s behavior? Which aspects of system performance can AUTOGNOSTIC modify?

ROUTER and KRITIK2 were the first two systems integrated with AUTOGNOSTIC and they provided the test-beds for investigating these issues. ROUTER (Goel & Callantine 1992) is a sophisticated and yet simple path planner, that uses multiple reasoning strategies and different types of domain knowledge to accomplish its task. Its knowledge about its domain includes a hierarchically organized model of its micro-world, and an episodic memory of previous path-planning problems
and their solutions. It also has some simple learning capabilities, since it extends its path memory by storing the result of each new problem-solving episode. KRITIK2 is an adaptive design system (Stroulias et al. 1992; Goel & Stroulias 1997). KRITIK2, like ROUTER, has multiple types of knowledge about its domain, including an episodic memory of cases of known devices, models which explain the functioning of these devices, and semantic knowledge of the substances and the components that are available in its technological environment. KRITIK2 uses a model-based method for the overall adaptive design task it performs, however, it has several methods for modifying the known devices to accomplish new functions.

There were several reasons for our choice of ROUTER and KRITIK2 as the initial test-beds for AUTOGNOSTIC. Both ROUTER and KRITIK2 are quite sophisticated systems, with non-deterministic processes and quite elaborate domain knowledge. Therefore, their modeling in the SBF-TMK language was sufficiently challenging to evaluate its expressiveness. In addition, they are capable of exhibiting a range of behaviors. By designing potentially desirable variances of these behaviors, we were able to explore different types of desirable behavioral changes that a redesign process should be able to bring about on a system. In addition, this exploration resulted in the refinement of the requirements that AUTOGNOSTIC’s process should fulfill and of the adaptation capabilities that it should possess. Finally, from a pragmatic point of view, both ROUTER and KRITIK2 were developed by researchers following the task-structure methodology for system design, albeit independently of the AUTOGNOSTIC project. Consequently, their design was quite easy to understand and model, thus simplifying the initial step of the process of integrating them with AUTOGNOSTIC.

4.1 Expressiveness of SBF-TMK models

ROUTER and KRITIK2 are both complex problem solvers: ROUTER’s SBF-TMK model consists of 25 tasks and KRITIK2’s of 9. An effort similar to that of modeling ROUTER or KRITIK2, i.e., approximately two to five person-days, would suffice for modeling a wide range of other, equally complex, systems with a documented functional architecture.
Result 0.1: AUTOGNOSTIC’s SBF-TMK language is sufficiently expressive to describe the design of complex deliberative systems with well-defined functional architectures.

If, however, there is no documentation of the system architecture or if it is imprecise, then there may be several alternative task structures that can describe the system. Two models of the same system may differ between themselves in several dimensions. First, they may decompose the system differently and, therefore, they may identify different subtasks and methods. Or they may decompose it in the same manner, but to a different level of detail. Or alternatively, they may decompose it in exactly the same manner but specify the different elements (tasks and methods) differently, i.e., in greater or less detail.

Our experiments with different versions of ROUTER’s SBF-TMK model demonstrated that such differences in a system’s potential models have an impact on AUTOGNOSTIC’s ability to adapt the system in question. For example, if the model is imprecise and the tasks are characterized only with sparse functional semantics, then the blame-assignment method may fail to evaluate whether the feedback solution is within the class of solutions that the system can produce, and also it may fail to suggest changes. If the model is not detailed enough and it only identifies a few big complex subtasks, then blame assignment may fail to localize the potential cause of the failure, and consequently it may fail to suggest operational modifications since the elements under inspection will be large and complex. Finally, the blame assignment task can only localize the cause of the failure to some of the identified elements; therefore, if the decomposition that the model captures does not identify the “wrong” element, then blame assignment will fail.

4.2 Redesign problems and Adaptation tasks

With respect to the types of adaptations that AUTOGNOSTIC’s process should be able to perform to a system, our initial exploration of system failures and corresponding corrective redesigns led us to conclude that both task-structure and domain-knowledge adaptations may be necessary to change a system’s solution quality. A system may fail to produce solutions of the desired quality
because its operators are incorrect, that is, they are not designed for that purpose. For example, if the desired behavior is a generalization of the delivered behavior, the functionality of some of the system operators may have to be extended. Alternatively, the cause of the failure may be lack of domain knowledge. For example, a desired behavior may rely on some piece of domain knowledge originally unavailable to the system, and may become possible as soon as this knowledge is provided to the system. By testing AUTOGNOSTIC with a variety of different scenarios, in which ROUTER and KRITIK2 failed to deliver the desired solution, we became confident that

**Result 0.2:** AUTOGNOSTIC’s process is able to identify and address both task-structure and domain knowledge errors.

### 4.3 Specifying Behavioral Changes

As we have already discussed, the basis of AUTOGNOSTIC’s redesign process in changing a system’s behavior is the feedback from the system users exemplifying the behavior desired of the system. An alternative approach is to model the desired behavior at an abstract level and identify the inconsistencies between the model of the desired behavior and the model of the system (Benjamin et al. 1996; Fensel & Schoenegge 1997; Musen et al. 1993). We chose to characterize the desired behavior in terms of examples, as opposed to in terms of abstract models, because, in general, domain experts can come up with specific examples easier than they can formulate abstract models. Therefore a process relying on examples rather than on models could be more generally applicable. On the other hand, this approach implies that the focus of AUTOGNOSTIC’s redesign has to be a particular class of behavior modification, that of qualitatively modifying the type of a system’s solutions. This is not a critical limitation since any redesign, whether it is motivated by solution quality or process efficiency, must have some impact either to the overall or to the intermediate solutions of the system. However the implication is that, if the redesign only affects an intermediate solution, i.e., some piece of information produced by an internal task of the task structure, then the feedback must be an example of what the desired alternative for this intermediate solution would be. The initial experiments performed with AUTOGNOSTIC-ON-ROUTER and
AUTOGNOSTIC-ON-KRITIK2 provided sufficient evidence to support that

Result 0.3: AUTOGNOSTIC’s process is sufficient for improving the quality of a system’s solutions, towards the kind of solutions exemplified by the users’ feedback.

5 Evaluation of AUTOGNOSTIC’s Redesign Process

Based on the results of the first exploratory phase, our original research hypothesis was refined as follows:

Hypothesis 1: AUTOGNOSTIC’s failure-driven redesign process, based on the SBF-TMK model of a system, should enable the system to produce better solutions, i.e., solutions of similar quality to the ones provided by its users as feedback to AUTOGNOSTIC.

The next step then was to precisely evaluate the effectiveness of AUTOGNOSTIC’s process in redesigning a system to improve its solution quality. The original ROUTER was designed to deliver satisficing, not optimal, paths. If the above hypothesis were correct, then AUTOGNOSTIC should be able, given paths consistently optimal according to some criterion, to redesign ROUTER so that it delivers optimal paths.

A random sequence of 150 problems in ROUTER’s domain was generated, and the original ROUTER was presented with each one of them. Subsequently, three different sequences of 40 problems, randomly selected from the original set of the 150 problems, were generated. Each of these three sequences was presented to AUTOGNOSTIC-ON-ROUTER in the context of two experiments. The underlying optimality criterion was path length (the sum of the costs of the path segments) in the first experiment, and path simplicity (the number of different streets traversed by the path) in the second. For each of the training problems, if AUTOGNOSTIC-ON-ROUTER did not produce the shortest/simplest path, it was given this path as feedback. AUTOGNOSTIC then proceeded to adapt ROUTER, if possible, to produce the feedback. At the end of each of these 40-training-problems sequences, the redesigned ROUTER was presented with the original 150-problems sequence. The quality of the solutions paths before and after was compared. The results of these experiments are
reported in the Tables 2 and 3.

[Table 2 about here.]

[Table 3 about here.]

The organization of the tables is similar. Each table contains three rows, each one reporting on the comparison of the original ROUTER with the ROUTER that resulted from the adaptations during each of the three 40-problem training sequence. The second column reports the number of problems for which the solution of modified ROUTER was better than that of the original, where the third reports the number for which it was worse. The fourth and fifth columns report two statistical measures for the comparison, sign test and pair t-test correspondingly. The improvement of ROUTER’s solutions after its redesign by AUTOGNOSTIC is statistically significant in both experiments.

The above experiments with AUTOGNOSTIC-ON-ROUTER produced the following two results:

**Result 1.1:** Effectively redesigning a system to improve the quality of its solutions may require modifications to its domain knowledge as well as its task structure.

**Result 1.2:** In the AUTOGNOSTIC-ON-ROUTER experiments, the SBF-TMK model of the system under redesign., i.e., ROUTER, enabled the effective localization of the adaptable system elements and therefore the relatively fast convergence of the overall redesign process.

These experiments with AUTOGNOSTIC-ON-ROUTER brought to the foreground the important issue of convergence, i.e., the issue of whether the redesign process converges to a single good design or it keeps on making modifications, perhaps undoing earlier ones in the process. Our inspection of the modifications actually performed during the six redesign sessions of the above experiments revealed that modifications of the system’s task structure were rather rare events, while modifications of its domain knowledge were more often. This relative frequency difference could be attributed to the nature of ROUTER’s task, which is domain-knowledge intensive, that is, it depends greatly on its map of its environment. It is possible that another system, with a more computation intensive
process that, unlike ROUTER, does not need to maintain a model of the environment, would exhibit more task-structure adaptations. In addition, in all six experiments AUTOGNOSTIC modified essentially the same elements of ROUTER’s task structure. This indicates that, given a particular design of a system, there are some elements that are more flexible than others and these are the elements that are bound to get modified when new requirements are imposed on the system’s performance. Furthermore, the task-structure modifications occurred in different patterns in these six experiments. In the “simple paths” experiment they all occurred early on, where in the “short path” experiments they were more evenly dispersed among the 40 training problems. An explanation for that phenomenon could be that different “optimization” criteria are more or less “natural” to the current system’s design, and thus the redesign process towards “optimizing” them converges faster or slower correspondingly.

6 AUTOGNOSTIC’s Experiments in Robotics

The success of the AUTOGNOSTIC-ON-ROUTER experiments verified our hypothesis that AUTOGNOSTIC’s redesign process can adapt a system to deliver solutions of a desired quality, under the assumption that the system is faithfully modeled in the SBF-1MK language. At this point of our work, we decided to continue experimenting with AUTOGNOSTIC’s redesign process, with the objectives of, first, exploring the boundaries of the process ability to adapt a system, and second, of evaluating how realistic the overall methodology was. Robotics systems emerged as a natural class of systems with which to evaluate AUTOGNOSTIC. First, because they live in the real world and often face unforeseen changes in their environment and in the requirements on their behavior, adaptation is an inherent part of their behavior. Second, most robots exhibit both deliberative and reactive behaviors, thus imposing additional constraints on the performance of a reflective redesign process. Third, robots have to meet real-time constraints on their behavior, and consequently any adaptation process must also conform with these constraints, if it is to be applicable to this class of systems. Finally, from a pragmatic point of view, two robots, REFLECS and DAVID, were
readily available to us for experimentation. In contrast to the previous two systems, they had been developed under different methodological frameworks. Therefore, they provided good test-beds for evaluating the usefulness of SBF-TMK modeling of independently built systems, since neither shares AUTOGNOSTIC’s design and development principles.

The experiments with REFLECS and DAVID were once again exploratory research. Their goal was to evaluate the effectiveness of AUTOGNOSTIC’s process in a context more general than the one in which the original research hypothesis was evaluated, and thus explore the limits of its applicability. The specific issues they were intended to explore were the following:

**Issue 2.1:** Are there methodological differences in modeling in the SBF-TMK language robots, i.e., hybrid agents integrating reactive and deliberative behaviors, vs. modeling deliberative systems?

**Issue 2.2:** How might AUTOGNOSTIC’s process be used in the absence of explicit feedback from the environment?

**Issue 2.3:** How do the actual problem solving and the redesign process interact?

The rest of this section describes the two robotics systems, REFLECS and DAVID, and discusses the experiments that were conducted with them to investigate the above issues.

### 6.1 REFLECS

REFLECS was implemented in the context of the AuRA architecture (Arkin 1989). In this methodology, the robot’s reactive behavior is accomplished through the interaction of a set of perceptual and motor schemas. Each perceptual schema is linked to some robot sensor to perceive some aspect of the environment. Each motor schema is linked to one or more perceptual schemas, and produces a motion vector as a function of the sensory information provided by them. The actual motion of the robot depends on the synthesis of all the vectors produced by all the active motor schemas. Our experiment with REFLECS was in the context of the office-rearrangement task, a task conceived as part of the AAAI-93 robotic competition. For this task, the robot is placed in an office with several
scattered boxes in it, and it has to move them into a designated area of the office. For each box it had to bring to the designated area, REFLECS went to the designated location, planned its path towards the target box, reactively reached this location, grasped the box, and brought it back to the designated location following blindly the same path. The experiment we discuss in this paper involves the third of these tasks, namely the task of reaching an individual box. The SBF-TMK model of the process is diagrammatically depicted in Figure 1.

[Figure 1 about here.]

The figure depicts tasks as solid-line boxes, and prototypes as bold solid-line boxes. Flow of control is represented by bold, solid arrows. Dashed boxes, indexed with dashed arrows by tasks, represent methods and they include the method’s subtasks. Finally, ellipses, attached to task boxes, represent operators accomplishing the leaf tasks of the task structure. The conditions and semantics of some of the tasks are shown as annotations at the low-right corner of the task box. The relations, to which these annotations refer, are part of the system’s domain knowledge.

In this figure, we see that the get-to-box task is accomplished by a method, which repeatedly invokes the step-in-get-to-box task. This task is accomplished by the interaction of four schemas. REFLECS uses the read shaft-encoder and read lasers perceptual schemas, to infer its position at each point in time and the positions of the static objects in its immediate environment. Two motor schemas, i.e., move-to-goal and avoid static-object, are also active. The first one uses the robot’s current position and the position of the target box to produce a vector pushing the robot towards the box. The second one uses the lasers’ readings and the robot’s current position to produce a vector repulsing the robot from the static objects around it. The actual motion of the robot is then decided by the synthesis of these two vectors, by the operator synthesize. The cycle stops when the shaft encoder shows that the robot has reached its goal, that is, when the robot is within a small radius from the target box.

3This experiment was run in simulation mode.
6.2 DAVID

DAVID (Strouila et al. 1997) was developed as a service robot for office-logistics tasks, such as delivering printout and mail, and collecting and emptying waste baskets. It has a variety of separate modules, each one accomplishing a separate functionality for perceiving, planning, moving, or manipulating the environment. The experiment we discuss in this paper involves DAVID’s waste-basket-collection task, the task structure of which is shown in Figure 2.

[Figure 2 about here.]

To collect a waste basket from an office, DAVID first must locate where exactly it is, then it has to pick it up, go to the collection location and put it down. To locate the basket, DAVID first plans a path to search effectively the office in question. This is accomplished by an operator that implements a heuristic art-gallery-planning search. This task produces a path that consists of a minimal set of observation points which suffice for observing the complete free space in an area. Having planned such a path, DAVID repeatedly selects a point from this path, goes there, and looks around to find the basket. If the basket is found, then DAVID docks it, thus completing the locate basket task. Otherwise, it proceeds to select another observation point from its art-gallery path. As soon as DAVID has completed the locate basket task, it can then proceed to pick up the basket, go to the collection location, and put it down there, thus completing its basket collection task.

6.3 Modeling Robots in SBF-TMK

Neither REFLECS’ nor DAVID’s development followed the task-structure methodology, thus making the problem of their modeling in SBF-TMK more challenging. On the other hand, their design architectures shared a lot of interesting features in common. First, they are both hierarchical with well-defined primitive elements at the bottom layer and well-defined composition rules. So for example, in the case of REFLECS, the leaf tasks are the functional abstractions of the schemas and their composition is based on the rule that perceptual schemas feed information to motor schemas.
which all feed information to the final synthesis schema. In the case of DAVID, the leaf tasks are the functional abstractions of its individual modules and their composition follows the high level control cycle of perception, planning, and action (i.e., motion and manipulation). This made the modeling of their task structures quite intuitive, in spite of not being explicitly designed within the task-structure methodology, and gave us more confidence in the expressiveness of the SBF-TMK language.

More generally, it led us to the conclusion that as long as the system’s internal processing can be viewed as a composition of well-defined elements, each of them with well-defined semantics, then this system can be quite faithfully modeled in the SBF-TMK language. In contrast, spreading-propagation type processing could not be modeled in SBF-TMK reasonably well, due to the intrinsically flat nature of the system’s elements and the large number of interactions among them. Once an SBF-TMK model of a system is constructed, then this system can potentially benefit from AUTOGNOSTIC’s redesign process. Whether this will actually happen depends on whether the system’s overall original design matches too closely its original requirements or whether it is to some degree under-specified, so as to allow for meeting potential evolutions of these requirements.

A second common feature in the models of the two robots was the fact that the concept of prototypes were used more extensively in their SBF-TMK models. As can be seen in Figures 1 and 2 there are quite a few prototype tasks in both these task structures: two out of nine in REFLECS and nine out of twenty one in DAVID, as compared to seven out of twenty five in ROUTER and none out of nine in KRITIK2. Deliberative problem solvers usually possess task structures that are highly tailored to their specific overall tasks. There are substantial differences among their high-level tasks and therefore there is a lot of variety in their task structures that consist mainly of compositions of individual tasks. In contrast, robots usually rely on a basic set of perceptual, planning and motion capabilities to accomplish their missions. The variety of their behaviors is due mainly to the dynamic nature of their environment, which gives rise to the need of appropriately configuring their basic set of functionalities. This difference in the character of problem solving of these two classes of systems results in the different character of their SBF-TMK models, i.e., method-based
task structures in deliberative problem solvers vs. configurations of prototype instances in robots. Note that this observation that arose from the modeling of the two robots enabled us to better define our modeling methodology. However, in spite of the differences in the character of the robotics SBF-TMK models as compared to the models of ROUTER and KRITIK2,

**Result 2.1:** as long as a system’s internal processing can be viewed as a composition of well-defined elements, each of them with well-defined semantics, then this system can be modeled in the SBF-TMK language.

### 6.4 Redesign in the Absence of Feedback

Another interesting difference between deliberative problem solvers and robots that we discovered in the course of our experimentation with REFLECS and DAVID was that, in the robotics domain, there are no domain experts to provide feedback on the desired behavior of the failing system. This fact poses a challenge to AUTOGNOSTIC’s redesign process that uses external feedback. As we have already mentioned in Section 3, AUTOGNOSTIC recognizes that the system is failing based either on external feedback or on the existence of problematic patterns in the systems behavior. While the vast majority of the ROUTER and KRITIK2 experiments used external feedback, the robotics experiments relied exclusively in this latter capability of AUTOGNOSTIC, and they demonstrated that

**Result 2.2:** given that the system’s design is redundant, that is, the system has multiple functionally similar modules, in the absence of feedback, AUTOGNOSTIC’s redesign process can be used for run-time reconfiguration of the system, in a manner that best enables appropriate behavior in its environment.

#### 6.4.1 Failure-Driven Self-Adaptation

When REFLECS was presented with the get-to-box task, it got close to the box and then it started oscillating in small steps around it. This was due to the avoid static-object schema, which repulsed it from the target box, at the same time when the move-to-goal schema pushed it towards it.
AUTOGNOSTIC’s monitoring recognized the lack of progress in REFLECS’ oscillatory motion, and suspended the task execution to assign blame for it. The detection of the failure is made possible by the knowledge captured by the SBF-TMK model of the system’s processing. First, based on the functional semantics of each task, AUTOGNOSTIC is able to evaluate whether the output of each step is correct by testing whether it verifies the semantics or not. The conditions of the step-in-get-to-box task specify that, this task is invoked as long as REFLECS has not reached the box, and its semantics specifies that, the motion vector it produces as output should be of magnitude significantly greater than zero. This semantics fails as REFLECS gets close to the box.

Alternatively, the SBF-TMK model provides yet another mechanism for detecting lack of progress, based on the causal interactions of the different subtasks in the system’s processing. By specifying move-to-goal as a subtask of the method accomplishing the step-in-get-to-box task, the model makes explicit that each instance of the former task in the system’s trace is performed in service of the latter. Therefore, when the exact same step, i.e., the move-to-goal subtask with the same input, is invoked in two consecutive cycles in service of the same step-in-get-to-box task, AUTOGNOSTIC recognizes that a task recurs and no progress is being made.

In its effort to assign blame for the synthesized vector of zero magnitude, AUTOGNOSTIC establishes that the synthesize task, responsible for producing the vector in question, was not accomplished in the last cycle, because the condition of its invocation was not met. Its input vectors were of opposite direction and same magnitude, and this task is not meaningful under such conditions. Thus, AUTOGNOSTIC infers as a potential cause for the failure the over-constrained conditions of the applicability of the synthesis task. As possible modifications, it suggests first, the replacement of the task with another which could be invoked even when the current one would not, or second, the falsification of the condition which prevents this task from being invoked.

The first modification would require the existence of a task, which would produce a vector even with opposite vectors as its input. The second modification requires that the information on which the synthesize task condition applies, i.e., the input vectors, has different values. The first modifica-
tion can be implemented by replacing *synthesize* with a task that, under these conditions, produces a “noise” vector to move the robot randomly out of the local minimum in which it is trapped. The second modification can be accomplished by replacing some of the tasks that produce the input of *synthesize*. It is noteworthy that when the designers of REFLECS encountered this problem they considered the former modification but they preferred to use the second one and developed an alternative *avoid* schema to replace *avoid static-object*.

AUTOGNOSTIC, given the knowledge about the existence of an alternative *avoid* schema, also selects the latter modification. The *avoid known-obstacle* schema does not avoid all static objects in the robot’s environment, but only the ones that are categorized as obstacles in its knowledge base. Notice that it is the *prototype-instance* relation among tasks that enables AUTOGNOSTIC to redesign REFLECS’ task structure in this experiment and correct its behavior. If AUTOGNOSTIC does not know about such an alternative *avoid* schema, then it can only propose the former modification.

### 6.4.2 An Experiment on Flexible run-time Control

Our experiments with DAVID revealed another style of adaptation that AUTOGNOSTIC can perform, that is, run-time task configuration. As we discussed in Section 6.3, and as can be seen from Figure 2 the decomposition of *go-to collection location* is based on instantiations of prototype tasks. This type of SBF-TMK model constitutes a compact description of a wide range of possible behaviors, each one applicable under different environmental conditions. AUTOGNOSTIC’s role in its integration with DAVID was to flexibly control DAVID’s behavior at run-time, so that it exhibits the behavior most appropriate for its current information state and environmental conditions. Let us discuss how.

The first time when DAVID has to go to a particular location is after it has planned a path to search the room and has selected to visit the first observation point in this path. Until that time, it has never before established its own position, therefore it cannot instantiate the *odometry-based self-localization* prototype, since this requires that its odometry sensors have already been initialized. On the other
hand, it knows in which area it is, and it can therefore instantiate the *landmark-based* prototype task, if there are landmarks at known positions in this area, or the *local-positioning-system-based* prototype task, if there are emitters in this area. If both conditions are met, AUTOGNOSTIC may invoke one of the prototypes at random. In addition, if one of the applicable prototypes fails to deliver the necessary information, i.e., DAVID’s current position, AUTOGNOSTIC will continue invoking the rest of them until one is successful.

Having established its current position, DAVID has to plan a path to its destination and move towards it. The former subtask can be an instantiation of either the *qualitatively plan* or the *latombe-like plan* task. The former generates a qualitative path from one area to another, and is applicable when DAVID does not have a specific point as initial or destination location but rather a general area. The latter produces a sequence of points in a metric coordinate system, and requires that both DAVID’s initial and destination location are specific points. Thus, while DAVID is searching for the waste basket in an office going from one observation point to another, AUTOGNOSTIC instantiates the latter prototype. When, however, DAVID has picked up the basket and needs to go to a collection area, with no particular location specified, the former prototype can be instantiated.

Finally, after having planned a path, DAVID has to execute it. Both *potential-fields-based-motion* and blind motion tasks require a quantitative path as input, where the *fuzzy-logic-based-motion* task can be instantiated with either a quantitative or a qualitative path. So the instantiation of a move prototype depends on which planning prototype has been already instantiated. Although there has been a lot of work on task-based run-time robot configuration, a notable example being (Firby 1989), this experiment demonstrated that AUTOGNOSTIC’s process, originally conceived for failure-driven redesign, could also play the same role.

### 6.5 Interleaving Problem Solving and Redesign

The last issue that arose directly from the reactive nature of a robot’s behavior is the interaction between the redesign and the actual problem-solving process. In AUTOGNOSTIC’s integration
with ROUTER and KRITIK2 the monitoring process and the reasoning process were completely integrated (see discussion in Section 3). In REFLECS, AUTOGNOSTIC’s reflection and the robot’s processing are two independent processes, but they are still synchronized at each reactive cycle. When there is a problem the reflection process interrupts the robot and takes over. This synchronization incurs a high overhead on the robot’s performance.

An alternative integration mechanism, explored in a subsequent REFLECS experiment (Goel et al. 1997), is to let the two processes run in parallel, and have monitoring done externally. That is, the deliberative reflection process does not monitor the actual processing of the reactive robot, but rather the trace of its progress towards the goal. Once a failure pattern, in the form of a behavioral cycle, is recognized, the reflection process and the internal monitoring kick in. The task is then performed in “slow motion,” and the trace of the problem solving is then examined by the deliberative reasoner in order to do blame assignment and reconfiguration. After the problem is fixed, the system returns to external monitoring to provide fast response.

To compare the advantages of internal versus external monitoring, we conducted five experiments with behavioral cycles in randomly generated worlds with fixed starting and goal positions. Twenty obstacles were generated for each experiment. The x and y coordinates of the obstacles were uniformly distributed random numbers between 0 and 64. The radius of the obstacles were uniformly distributed random numbers between 1 and 5. An obstacle was deleted if it overlapped the start or goal locations. Table 4 summarizes the efficiency of processing using internal and external monitoring. The time spent on the task is the time from the first simulation iteration to the iteration when the robot reaches the goal. It was registered using \texttt{time()} function in the simulation program.

[Table 4 about here.]

A paired t-test shows that on average, the time taken for a step using external monitoring is 0.08 seconds, and the average time for a step using internal monitoring is 3.44 seconds. The t value for the paired t-test is 19.2, significantly above the 99.5% confidence level. This experiment demonstrated that
Result 2.3: problem-solving and redesign can be interleaved in two modes: by monitoring the internal process and by vs. observing the external behavior; for systems with real-time constraints the latter mode incurs less overhead and is therefore preferable.

7 Related Research

Our work on AUTOGNOSTIC lies within the areas of knowledge-level system modeling, reflection, and system redesign and reconfiguration.

Research in knowledge-level modeling was originally motivated by the problem of knowledge acquisition. Chandrasekaran (Chandrasekaran 1983) proposed the concept of “generic tasks”, and identified a set of such tasks permeating several fields of problem-solving activity, all of which share a standard pattern of information and control flow although their knowledge ontologies may differ. At around the same time, Clancey (Clancey 1985) analyzed in detail the inference structure of heuristic classification underlying a series of expert systems, and McDermott (McDermott 1988) started developing a series of knowledge-acquisition tools, based on the concept of “role-limiting problem-solving methods” that can be used to perform families of tasks by abstracting the control knowledge from domain-specific details. Finally, in a similar effort, Wielinga and his colleagues (Wielinga & Breuker 1986; Wielinga et al. 1992) described a four-layer model of expertise, KADS, which also was based on the distinction between high-level domain-independent strategic and control knowledge and domain-specific ontologies.

All these lines of work focus on the identification of a finite set of tasks with generic, domain-independent control structures. These tasks are thought to be applicable across domains and their instantiation in any particular domain relies on the existence in this domain of specific types of knowledge that can fill the roles that the tasks’ control assumes. In a related vein, Chandrasekaran (Chandrasekaran 1989) and Steels (Steels 1990) extended and evolved the ideas of the generic-tasks and KADS frameworks to develop frameworks for describing system-specific task structures, which enabled the specification of domain-specific, non-generic control structures, in addition to
the generic ones.

All these frameworks have been used in several different contexts and have each spawned a family of other projects, several of which have addressed redesign-related issues. In general, one can identify two broad lines of issues under investigation: design of specific systems by adaptation of generic PSMs, and redesign of specific systems by reconfiguration of their basic task-structure elements. Within the KADS framework, Fensel and Schoenegge (Fensel & Schoenegge 1997) have focused on the automated derivation of task-specific PSMs from generic ones by successive transformations. A similar investigation has been also pursued by the Protege project (Musen et al. 1993). Among the projects deriving from the KADS methodology, Reflect and Desire have addressed the issues of system reconfiguration. The Reflect project (Harmelen et al. 1992) investigated the use of reflection in developing flexible knowledge-based systems that could reason about and potentially adapt the ways in which they can employ their knowledge. A subsequent investigation by Teije focused specifically on the automated reflective configuration of diagnostic PSMs (Teije 1997). In the context of the Desire project (Brazier, Jonker & Treur 1998) Treur, Brazier and their colleagues have been investigating the automated redesign of compositional systems, such as cooperative multi-agent systems.

These projects and AUTOGNOSTIC share a common methodological approach: they all have developed knowledge-level modeling frameworks and corresponding reflective methods for reasoning and, to some extent, adapting the design of systems. The fundamental difference distinguishing AUTOGNOSTIC from Reflect and Desire is that, not only does it reconfigure the basic elements of a system’s task-structure, but also it redesigns these elements. The most distinguishing characteristic of AUTOGNOSTIC’s redesign process, as compared with that of Protege and the work of Fensel and his colleagues, is that it is based on the SBF∞-TMK model of the specific system instead of a model of a generic PSM.
8 Discussion and Conclusions

In this paper, we reported on a sequence of experiments we conducted with AUTOGNOSTIC, a shell for system modeling and redesign. Taking a step back to reflect on our evaluation methodology and agenda, we would like to make a few comments. We described three research phases: the first and third ones were mainly exploratory where the second was oriented towards evaluation.

The main objective of the first phase was to explore the range of modifications that AUTOGNOSTIC’s process could support, and to fine-tune the design of the SBF-TMK language in order to enable AUTOGNOSTIC to accomplish these modifications. In that sense, the results of this first phase were not particularly surprising or original. They were basically in conformance with similar results reported by the KE community. In the context of our research program, they were useful though in that they established a baseline for continuing our research on AUTOGNOSTIC and they led us to define precise research problems for which well-defined testable hypotheses could be formulated.

The second phase of experiments with ROUTER was focused exactly on the evaluation of a specific hypothesis. It did that by measuring the effectiveness of AUTOGNOSTIC’s redesign process as it had matured through the experimentation of the previous exploration phase. The results of this phase characterize very precisely the effectiveness of AUTOGNOSTIC’s process in redesigning a system towards accomplishing tasks similar to, but slightly different from, the ones it was originally designed for.

The third phase, consisting of the REFLECS and DAVID experiments, was again exploratory. Its objective was to challenge AUTOGNOSTIC’s process by applying it to a novel class of systems, developed by researchers unaware of the design principles shared by the KE community. Because of that objective, the results of these experiments were quite more interesting than these of the first exploratory phase. The novelty of the class of object systems enabled us to better understand the boundaries of the SBF-TMK expressiveness and also to recognize the run-time implications of our
original AUTOGNOSTIC-on-object-system integration mechanism.

These two phases of research, exploration and evaluation (Cohen 1995), are closely intertwined. Exploration investigates interesting issues related to the general research problem, and enables the researchers to make further methodological commitments, to refine their hypotheses and to develop the tools to test them. Subsequent evaluation steps investigate the degree to which the developed tools confirm the hypotheses and identify new challenges for them. In the AUTOGNOSTIC’s experiments, the first exploratory phase refined the original high-level hypothesis of “redesign for behavior modification” to “redesign for solution quality improvement”. In turn, challenging the generality and realism of AUTOGNOSTIC’s redesign process with the robotics experiments of the third exploratory phase brought about the issue of the efficiency and performance of this process. This issue never arose in the experiments with ROUTER and KRITIK2, since, in contrast to robots which act in real time, these systems do not face any time constraints.

As a commentary on our methodology, we should mention the dimensions we considered of importance in the evaluation of our research. These are:

- effectiveness
- generality
- realism

**Effectiveness:** Clearly, measurable evidence of a method’s effectiveness is a prerequisite for having a major impact within the research community and also for advancing the state of the art. To evaluate effectiveness, the precise formulation of the research hypothesis is necessary. The second experiment phase played that role in our research program.

**Generality:** Even if a method is shown to be effective, if it is only applicable in a limited domain, its utility is limited. To evaluate AUTOGNOSTIC for generality, we integrated AUTOGNOS-
TIC with many and diverse systems. Our choice of systems was to some degree intentional and
to some degree opportunistic. We feel that these four systems are interesting examples from the
systems space and together provide an interesting enough coverage. They gave us a better insight
on the class of systems to which AUTOMNOSTIC’s redesign is applicable. For a sharper character-
ization of the boundaries of the SBF-TMK language expressiveness, a much more sophisticated
characterization of the dimensions of the systems space would be required. And for complicated
tasks, such as knowledge engineering and adaptive design, it is not clear that all the relevant dimen-
sions can be identified a-priori, and perhaps a quality-function-deployment methodology (Akao
1990) could be deployed to identify the more important dimensions with respect to the goals of the
language/process under investigation.

**Realism:** Very sophisticated methods can often be developed for problems formulated in a man-
ner too narrow to be realistic. We believe that, in addition to formulating an evaluation plan early
on in the process of designing the method in question, it is very useful, at the end of the develop-
ment phase, to search again about new evaluation scenario, as independent of the method design
and development as possible. The REFLECS and DAVID experiments were such choices for AU-
TOGNOSTIC’s evaluation.

**Acknowledgments**

An earlier version of this paper was presented in the plenary session of the track on “Evaluation
of Knowledge Engineering Techniques” at the Eleventh Workshop on Knowledge Acquisition,
Modeling, and Management (KAW'98), held in Banff, Canada in April 1998. We would like to
thank the anonymous reviewers of this earlier version as well as many of the participants to the
workshop for their insightful comments on our work.
References


List of Figures

1. REFLECS’ task structure. ........................................... 35
2. DAVID’s task structure. ........................................... 36
FIGURE 1: REFLCS’ task structure.
FIGURE 2: DAVID’s task structure.
List of Tables

1. Elements of the SBF-TMK language .................................................. 38
2. First Experiment: Short paths .......................................................... 39
3. Second Experiment: Simple paths .................................................. 40
4. Internal vs. External Monitoring ..................................................... 41
### Table 1: Elements of the SBF-TMK language.

- **Task** $T := (i, o, \{p\}, by/op, c, s)$
  - $i$ input information, $o$ output information, $p$ prototype task of which the current task is an instance, $by$ the methods for accomplishing the task, $op$ an operator that accomplishes the task (if it is a primitive design element of the system), $c$ the condition when the task should be performed, and $s$ the task’s functional semantics, partially describing the transformation between its input and output.

- **Method** $M := (t_{up}, c, t_{sub}, ctrl)$
  - $t_{up}$ the task that the method accomplishes, $c$ the conditions when it is applicable, $t_{sub}$ the subtasks that it sets up for $t_{up}$, and $ctrl$ control relations, such as precedence, parallelism and repetition, over the subtasks.

- **Information** $I := (w_{o}, t_{in}, t_{out})$
  - $w_{o}$ the domain concept of which the information is an instance, $t_{in}$ the tasks that consume the information, and $t_{out}$ the tasks that produce it.

- **Concept** $Cnpt := (d, attrs, id)$
  - $d$ the domain of the concept’s values, $attrs$ the concept’s attributes, $id$ a predicate that evaluates the identity between two instances of the concept.

- **Relation** $DR := (i, o, tt, p, ip, indx)$
  - $i$, $o$ the domain and range of the relation respectively, $tt$, $p$, and $ip$ the relation’s truth table, predicate and inverse predicate.

- **Constraint** $DC := (i_{f}, then, p)$
  - $i_{f}$ and $then$ are the relations to which the constraint refers, and $p$ is the predicate that the elements of these relations should conform with.
<table>
<thead>
<tr>
<th># sequence</th>
<th># of Problems with Shorter paths</th>
<th># of Problems with Longer paths</th>
<th>Sign test</th>
<th>Paired t-test results</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence 1</td>
<td>55</td>
<td>13</td>
<td>$1.3 \times 10^{-7}$</td>
<td>0.0002 (t=3.88)</td>
</tr>
<tr>
<td>sequence 2</td>
<td>53</td>
<td>14</td>
<td>$8.8 \times 10^{-7}$</td>
<td>0.0000 (t=5.17)</td>
</tr>
<tr>
<td>sequence 3</td>
<td>44</td>
<td>30</td>
<td>0.06</td>
<td>0.14 (t=1.49)</td>
</tr>
</tbody>
</table>
### TABLE 3: Second Experiment: Simple paths

<table>
<thead>
<tr>
<th># sequence</th>
<th># of Problems with Simpler Paths</th>
<th># of Problems with more Complex Paths</th>
<th>Sign test</th>
<th>Paired t-test results</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence 1</td>
<td>60</td>
<td>10</td>
<td>$4 \times 10^{-10}$</td>
<td>0.0000 (t=5.92)</td>
</tr>
<tr>
<td>sequence 2</td>
<td>50</td>
<td>22</td>
<td>$6.4 \times 10^{-4}$</td>
<td>0.25 (t=1.15)</td>
</tr>
<tr>
<td>sequence 3</td>
<td>56</td>
<td>15</td>
<td>$5.2 \times 10^{-7}$</td>
<td>0.0000 (t=4.82)</td>
</tr>
<tr>
<td>Exp. Number</td>
<td>Steps Taken</td>
<td>Time with External Monitoring</td>
<td>Time with Internal Monitoring</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------------------------------</td>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>93</td>
<td>8</td>
<td>366</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>71</td>
<td>5</td>
<td>231</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>84</td>
<td>6</td>
<td>318</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>65</td>
<td>6</td>
<td>206</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>5</td>
<td>184</td>
<td></td>
</tr>
</tbody>
</table>