Task Acquisition with a Description Logic Reasoner

RR-95-0
Research Repo

M. Buchheit, H.-J. Bürckert, B. Hollunder,
A. Laux, W. Nutt, M. Wójcik

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report

RR-95-04

Task Acquisition with a Description Logic Reasoner

M. Buchheit, H.-J. Bürckert, B. Hollunder,
A. Laux, W. Nutt, M. Wójcik

May 1995

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry of Education, Science, Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland

Director

Task Acquisition with a Description Logic Reasoner

M. Buchheit, H.-J. Bürckert, B. Hollunder,
A. Laux, W. Nutt, M. Wójcik

DFKI-RR-95-04

A version of this paper will appear in the Proceedings of the 19th German Annual Conference on Artificial Intelligence, KI-95.

This work has been supported by a grant from The Federal Ministry of Education, Science, Research and Technology (FKZ ITWM-9201).

ISSN 0946-008X

Task Acquisition with a Description Logic Reasoner

Martin Buchheit Hans-Jürgen Bürckert Bernhard Hollunder
Armin Laux Werner Nutt Marek Wójcik*

German Research Center for Artificial Intelligence - DFKI GmbH
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
e-mail: tacos@dfki.uni-sb.de, fax: (+49 681) 302 5341

Abstract

In many knowledge based systems the application domain is modeled in an object-centered formalism. Research in knowledge acquisition has given evidence that this approach allows one to adequately model the conceptual structures of human experts. However, when a novice user wants to describe a particular task to be solved by such a system he has to be well acquainted with the underlying domain model, and therefore is charged with the burden of making himself familiar with it. We aim at giving automated support to a user in this process, which we call *task acquisition.*

This paper describes the TACOS system, which guides a user through an object-centered domain model and gives support to him in specifying his task. A characteristic of TACOS is that the user can enter only information that is meaningful and consistent with the domain model. In order to identify such information, TACOS exploits the ability of a description logic based knowledge representation system to reason about such models.

*on leave from the Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Contents

1 Introduction

In many areas of Computer Science and Artificial Intelligence, like Programming, Databases, and Knowledge Based Systems, formalisms for modeling an application domain share a similar view of the world. They perceive a universe as consisting of objects, which are grouped into hierarchically organized classes and linked by attributes. The classes and attributes are further specified by integrity constraints and rules that can be expressed in some fragment or variant of predicate logic.

In particular, in knowledge acquisition research, which has the goal of developing knowledge representation schemes that are capable of capturing the conceptual structures of human experts, such an object-centered approach to domain modeling has proven useful. Meanwhile, a number of tools have been built that support knowledge acquisition in this paradigm (see [6, 7]).

As a result, at the core of many knowledge based systems there is an object-centered domain model. It might be implemented on different kinds of platforms such as object-oriented databases, knowledge representation systems based on Descriptions Logics (DLs), like LOOM [9], CLASSIC [11] and KRIS [1], or even knowledge acquisition tools that allow one to execute the elicited knowledge, like KSSn [6] or EXPECT [7].

The advantage of rich domain models is that expert knowledge can be represented adequately. However, a user who wants a particular problem to be solved has to know the model well in order to describe his case in a meaningful way. Such a task description for the system will typically consist of a number of objects in the model and relations between them. Thus, a novice user of a knowledge based system is charged with the burden of making himself familiar with the underlying model and to figure out what kind of input is expected by the system.

In analogy to the notion of knowledge acquisition, we call *task acquisition* the problem of giving automated support to such a non-initiated user in specifying his case. Task acquisition can be seen as a special case of knowledge acquisition as it deals with eliciting knowledge about a particular case to be solved. There is a difference, however, in that the terms in which a task is to be expressed are already fixed and have to be communicated to the user.

In this paper we present the system TACOS (= Task ACquisition for Object-centered Systems), which guides a user through an object-centered domain model and gives support in querying it and populating it with relevant objects. A tool like TACOS can either function as a frontend to a knowledge based system or serve as a platform for prototyping simple such systems. Because of its support for describing objects in terms of a rich object-centered domain model it can be employed for filling and maintaining a large fact base, but also for specifying a particular task. In this paper, we concentrate on the second usage because all services of TACOS come in useful.

The rest of the paper is organized as follows. In the next section we give an

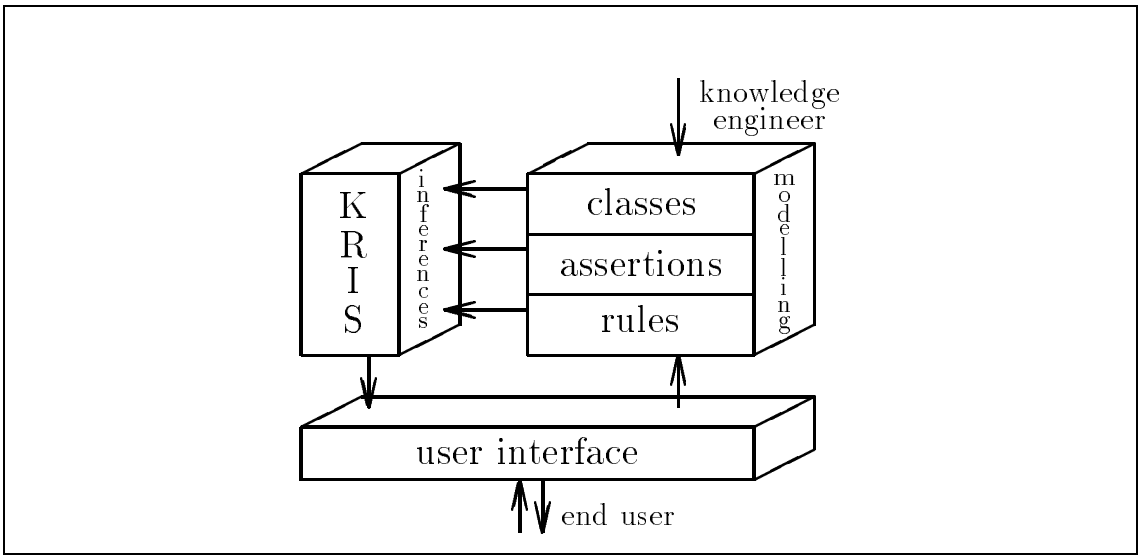

Figure 1: The architecture of TACOS.

overview of the system. Sections 3 to 5 describe the main building blocks, which are the modeling and the inference component, and the user interface. In Section 6 we discuss further application scenarios and possible extensions. Section 7 concludes.

2 A Bird's Eye View of TACOS

We now sketch the components of the system and how they contribute to the overall functionality.

2.1 The architecture

TACOS consists of three major components (see Figure 1): a domain modeling component, an inference component, and a user interface.

With the *modeling component* one sets up a model of the problem domain. A domain model consists of three parts: (1) an ontology describing the relevant *classes* with their attributes as well as simple integrity constraints that are to hold between them, (2) *assertions* about objects and relationships between them, and (3) a set of monotonic *inference rules*, by which additional relationships between objects can be derived. We also refer to ontology and assertions as the *static part* of a domain model and to the rules as the *dynamic part*. The language of the static part contains the essentials of object-centered modeling formalisms like object-oriented data models, frame systems, or DLs. The rule language is Turing complete and thus sufficiently expressive to build at least small applications in TACOS itself.

Deductions from the statements in the domain model are drawn in the *inference*

component, which is realized by the DL based knowledge representation system KRIS [1]. With the help of the inference component, the *user interface* produces menus suggesting assertions which the user can enter without compromising the consistency of the static part.

2.2 The functionality

In our view, a task is described by objects that are—not necessarily completely—specified in terms of the domain model. A knowledge based system responds by either (1) completing the specification, (2) generating new objects, or (3) producing output on windows controlled by itself. A user is assisted by TACOS in specifying objects through a menu-based interface. The interface displays the current information about the known objects in terms as specific as possible and suggests possible inputs in its menus. The suggestions encompass only meaningful items in the sense that adding them does not lead to inconsistencies with respect to the static part of the domain model. Obviously, due to the expressivity of the rule language it is impossible to restrict the suggested input further so that also inconsistencies produced by rules are prevented. However, we assume that in general the rules are written in such a way that this case does not occur. If it happens, earlier states of the session can be recovered with a backtracking mechanism.

The whole approach is flexible in that the appearance of the menus and the guidance during the acquisition process are completely derived from the domain model.

It is a characteristic of TACOS that only information can be entered that is meaningful and consistent with respect to the domain description. In order to identify such information, the domain model is translated into a KRIS knowledge base. Through its inferences KRIS is able to determine into which classes a given object can still be put and which constraints can be imposed on its attributes without compromising consistency. Moreover, it can find all objects that satisfy a given set of constraints.

For two reasons a system like KRIS is an appropriate inference engine for task acquisition. The first pertains to DL systems in general: Since the purpose of the system is the assistance in specifying a task, it is confronted with incomplete information. Such incompleteness is taken into account by the open world semantics of DLs. The second is more specific: In order to detect all possible inconsistencies, the reasoning process must not only be sound, but also be complete. There are only few DL systems with this property, and KRIS is one of them.

In the next three sections we will illustrate the languages, the components, and the services of TACOS with a running example taken from a scenario where a user specifies a transport task to be executed by a forwarding agency. We have implemented this scenario in TACOS at DFKI.

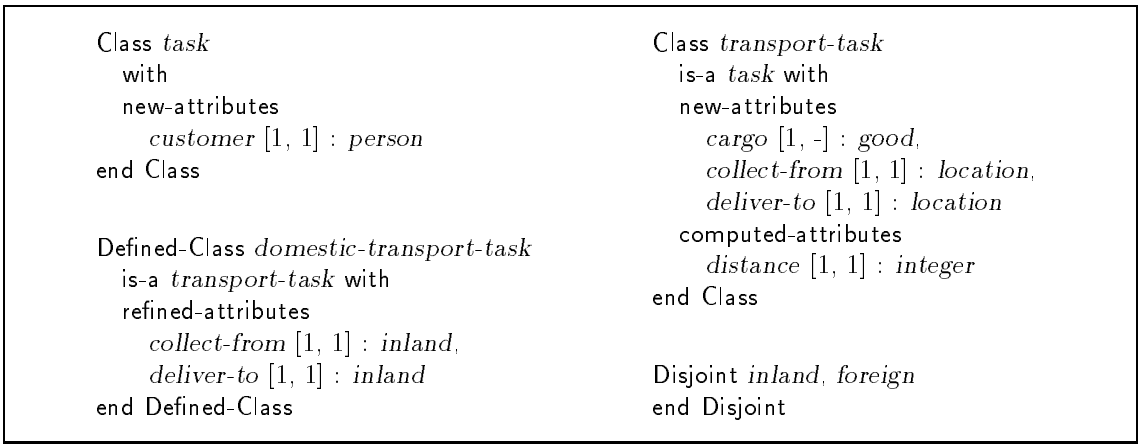

Figure 2: Classes in our forwarding domain.

3 The Domain Modeling Component

For each part of the modeling component (see Figure 1) TACOS provides a particular language: (1) a class description language, (2) an assertional language, and (3) a rule language.

With the *class description language* one fixes the structure of a problem domain. Basically, a class describes a set of objects in terms of more general classes and of attributes. Figure 2 contains the description of three classes in our forwarding domain. Classes come in two variants: basic classes and defined classes. The description of a *basic* class (indicated by the keyword Class) imposes *only* necessary conditions for class membership. So, the description of the basic class *task* says that for every task there is exactly one person which is the customer, or formally, if an object is an instance of *task*, then it must have exactly one filler of the attribute *customer*, and the filler must be an instance of the class *person*. The class *transport-task* is a specialization of *task*. Hence, *transport-task* inherits from *task* the attribute *customer*. In addition, a *transport-task* has the attributes *cargo*, *collect-from*, *deliver-to*, and *distance*, which are subject to the following constraints: (1) each *cargo* is a *good*, and there is at least one *cargo*, (2) there is exactly one *collect-from* and one *deliver-to*, which are *location*s, and (3) there is exactly one *distance*, which is an *integer*. The *distance* is computed by the system and cannot be input by the user. The computation is specified by rules (see below).

The class *domestic-transport-task* specializes *transport-task* in that the range restrictions for the inherited attributes are refined: for a *domestic-transport-task* the fillers for the two attributes *collect-from* and *deliver-to* must not only be *location*s, but additionally must be instances of *inland* which is a subclass of *location*. Since *domestic-transport-task* is a *defined* class (indicated by the keyword Defined-Class), the description specifies both necessary *and* sufficient conditions for class membership. This means that any object (1) which belongs to *transport-task*, and

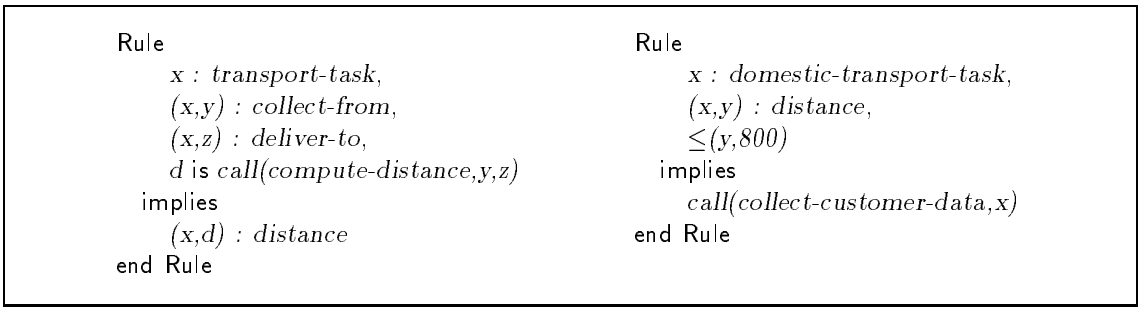

Figure 3: Two rules in our forwarding domain.

(2) whose fillers for *collect-from* and *deliver-to* belong to *inland*, is also an instance of *domestic-transport-task*. Defined classes can be seen as predefined queries or "views" [4], which help in structuring the domain. They can be combined to complex queries and provide macro to be used in the preconditions of rules.

In addition to the means for describing classes discussed so far, one can define *enumeration classes* (like the class *color*, containing exactly the objects *yellow*, *red*, and *blue*), declare a group of basic classes as *disjoint* (like stating that *inland* and *foreign* cannot have common instances), and specify that one basic class covers a group of other basic classes (like introducing *task* as the union of *transport-task* and *storing-task*). Furthermore, there are the built-in classes *string* and *integer*, as well as intervals of *integer*s.

In the *assertional language* one describes a given state of affairs in the problem domain by (1) inserting an object into a class (e.g., declare *my-order* as a *transport-task*) and (2) relating objects through attributes (e.g., make *Berlin* the filler of *deliver-to* for *my-order*). Usually, TACOS will start with an initial knowledge base (KB for short) containing facts about individuals that are relevant for the domain. In our forwarding example we assume that there are facts about cities such as $Bonn : inland$, $Berlin : inland$, $London : foreign$, $(Bonn, 300.000) : inhabitants$, etc.

In the *rule language* one can specify conditions for deriving new assertions, generating fresh objects, or calling functions in the host language (which is LISP). A rule consists of a *condition part* and a *consequence part*. As an example consider the rules in Figure 3.

Suppose that *my-order* is a *transport-task*, and that *Bonn* and *Berlin* are the corresponding fillers for *collect-from* and *deliver-to*. Then the first rule can be applied to compute *my-order*'s filler of the attribute *distance* as follows: as the first three conditions are entailed by the static part (if the variables x, y, and z are substituted by *my-order*, *Bonn*, and *Berlin*), the function *compute-distance* is called with the arguments *Bonn* and *Berlin*, and the resulting value, say *598*, is bound to the variable d; thus, the rule fires and the fact that *my-order*'s filler of *distance* is *598* is added to the current set of assertions.

The effect of applying the second rule is the execution of a function, which is

called for its side effects. As *my-order*, in our example, is a *transport-task* whose fillers for *collect-from* and *deliver-to* are instances of *inland*, it is recognized as a *domestic-transport-task*. Moreover, the filler of *distance* is not greater than *800*. Thus all preconditions are satisfied and the function *collect-customer-data* is called with argument *my-order*.

For class descriptions and assertions we assume a straightforward first order semantics, similar to that for DLs. Rules without function calls can be explained with the help of an epistemic operator (see [5]).

The class description language of TACOS captures essentially what can be expressed in common object-centered representation formalisms like frame languages, object-oriented data bases, and DLs. The rule language, however, goes beyond the rules generally available in data bases and DLs. Since it allows for the generation of fresh objects, it is Turing complete even if we ignore the integration with the host language.

4 The Inference Component

The purpose of the domain modeling component is to state facts about a problem domain. To draw conclusions from the represented knowledge we utilize the inference mechanism of KRIS. As a full-fledged DL system, KRIS provides three languages for the representation of knowledge: (1) a terminological language, (2) an assertional language, and (3) a rule language.

In the *terminological language* one can define concepts by means of complex expressions, which are built up from other concepts using constructs such as conjunction ($C \sqcap D$), disjunction ($C \sqcup D$), negation ($\neg C$), range restriction ($\forall R.C$), and cardinality constraints (($\geq n\ R$), ($\leq n\ R$), ($= n\ R$)). To see how classes in the domain modeling component are mapped to concepts, consider the class *transport-task*. Its corresponding concept is declared as

$$\begin{aligned} \textit{transport-task} \quad \dot{\sqsubseteq} \quad & \textit{task} \sqcap (\geq 1\ \textit{cargo}) \sqcap \forall \textit{cargo.good} \sqcap \\ & (= 1\ \textit{collect-from}) \sqcap \forall \textit{collect-from.location} \sqcap \\ & (= 1\ \textit{deliver-to}) \sqcap \forall \textit{deliver-to.location} \sqcap \\ & (= 1\ \textit{distance}) \sqcap \forall \textit{distance.integer}. \end{aligned}$$

While classes can be translated into concepts in a straightforward manner, the converse is not true, as the terminological language of KRIS contains a number of constructs that have no counterpart in TACOS. The *assertional language* and the *rule language* in KRIS are similar to the ones in TACOS, except that the former are based on concepts and the latter on classes. Like a domain model, a KRIS *knowledge base* comprises a static and a dynamic part (static and dynamic KB for short). The *static* KB consists of a set $\mathcal{T}$ of concept declarations and a set $\mathcal{A}$ of assertions, the *dynamic* KB consists of a set of rules.

Static KBs are interpreted under the standard DL semantics, which identifies them with a set of first order formulas (see [10]), and inferences are defined with respect to this semantics. Elementary inference problems are to decide, given a static KB, whether one concept subsumes a second, whether an object is an instance of a concept, and whether the KB is consistent at all. More complex reasoning services, like determining all subsumption relationships between the concepts in the KB (classification) and finding all memberships of objects in concepts (realization) are based on the elementary inferences.

When considering inferences, a characteristic comes into play that distinguishes DLs from databases: a KB is not viewed as determining a single structure, but as having—possibly infinitely many—different models: for inferences *all* first order models of the KB are taken into account. For this reason, we call this semantics an "open world semantics" as opposed to a "closed world semantics" admitting only a single model.

For the purpose of task acquisition a closed world semantics would be inappropriate because a task is not completely specified before the very end of a session and various completions are conceivable. In particular, it is not justified to conclude negative information from the absence of information, like a conventional database system would do. For example, the fact that up to a given moment no dangerous cargo has been specified for a transport task does not necessarily imply that all cargo will be safe. Thus, a task acquisition system has to take into account all possible completions in its reasoning process.

In almost all implemented DL systems, inferences are realized by sound but *incomplete* algorithms. This, for example, means that if a consistency checking procedure reports a static KB to be inconsistent, one can rely on this answer. However, if the procedure reports consistency, nothing can be concluded: the KB may or may not be consistent. For the use in TACOS, a DL system with incomplete algorithms is obviously insufficient since it could not prevent a user from entering inconsistent information. However, KRIS is an appropriate choice because its algorithms are sound *and* complete [2].

To achieve the functionality of TACOS, we have extended KRIS to perform additional reasoning services, which are based on elementary inferences, but which usually are not provided by DL systems. Among them are the following, where $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ is a static KB, C a concept occurring in $\mathcal{T}$, R an attribute, and a, b objects:

Possible concepts: C is a *possible concept* for a if $\langle \mathcal{T}, \mathcal{A} \cup \{a\!:\!C\} \rangle$ is consistent;

Possible range restrictions: C is a *possible range restriction* of R for a if $\langle \mathcal{T}, \mathcal{A} \cup \{a\!:\!\forall R.C\} \rangle$ is consistent;

Possible fillers: b is a *possible filler* of R for a if $\langle \mathcal{T}, \mathcal{A} \cup \{(a,b)\!:\!R\} \rangle$ is consistent;

Plausible fillers: b is a *plausible filler* of R for a if for all concepts D having

a counterpart in the class language of TACOS we have that $\langle \mathcal{T}, \mathcal{A} \cup \{(a,b):R\}\rangle \models b:D$ if and only if $\langle \mathcal{T}, \mathcal{A}\rangle \models b:D$.

Possible concepts, range restrictions and fillers for an object are computed when TACOS determines which information a user can enter safely. These inferences are based on consistency checks. The check for plausible fillers, however, is more involved. Intuitively, b is plausible if it already has all the properties—expressible as memberships of b in some D—that a filler of R for a is required to have.[1] Computationally, this check is realized by abstracting the constraints imposed on the fillers of R into a concept and retrieving its instances. Looking up plausible fillers is helpful when a user combines entering information with querying the domain model for interesting objects.

Rules in KRIS are handled analogously to rules in TACOS. The key inference for deciding whether a rule is applicable is the test whether an object is an instance of a concept. A set of rules is applied by executing each rule with all possible instantiations of its variables (see [8]).

5 The User Interface

The user interface assists a user who wants to specify a task in populating and querying a domain model. It has been designed so that it (1) guides the user during the acquisition phase, (2) passes the information acquired from the user to the domain modeling component, and (3) displays appropriate help texts whenever the user needs additional information (e.g., the description of objects or classes).

In the sequel we illustrate the first of these functionalities with an example session in our forwarding scenario. TACOS realizes this functionality by displaying the current information about objects, and by proposing to the user new pieces of information to add. In order to present only information as specific as possible and to offer all possibilities of entering compatible information, the user interface triggers inferences in KRIS.

A TACOS session starts with the main window of the system displayed on the screen (see Figure 4 in the appendix). From here the user proceeds by choosing an object—either a new one or one present in the domain model—for which he wants to add information.

[1] Of course, the definition depends on the language that D ranges over. For instance, if singleton concepts, inverse attributes, and existential quantification over attributes can be expressed in the language, we can conclude from $\{(a,b):R\}$ the membership $b:\exists R^{-1}.\{a\}$. Since such a membership cannot be concluded for objects c that are not R-fillers of a, in this case only objects that are already fillers are plausible fillers. Neither of these constructs is expressible in TACOS.

5.1 Object windows

When data for an object is retrieved from the KB an *object window* showing the information available about the object appears on the screen. An object window consists of an *object level* part and an *attribute level* part. The first contains the name of the object and a list of the (most specific) classes it belongs to; the second contains descriptions of all attributes of the object. Each class, attribute, and object name in the object window is mouse-sensitive. When clicking into it a pull-down menu pops up offering operations that can be applied to that entity. We explain the most important operations continuing the example from our forwarding domain. Suppose, we have started with the object *my-order* of the class *task*. Then the system displays the following information in an object window.[2]

object:	*my-order*		class(es): *task*		
attribute(s):	name	range	min	max	filler(s)
	customer	*person*	1	1	—

The object level. On the object level the only data that can be modified is the list of classes the object belongs to. The user can refine a class from the list and extend the list.

Refining a class means replacing it by a more specific one. To this end the user chooses an element from a list of classes offered to him by the system. The list is computed by KRIS, which tests for each concept corresponding to an immediate subclass of the class to be refined whether it is a possible concept for the object at hand (see Section 4). The support of KRIS allows the user to focus on a restricted set of relevant items and guarantees that only consistent information is entered. Assume there are the following subclasses of *task*: *transport-task*, *storing-task* (which are disjoint), and *single-task*, *recurring-task*. If we want to refine the class *task* of *my-order*, all of them are possible and the system offers the four of them. Suppose we choose *transport-task*. As additional attributes are declared on this class the system updates the displayed information:

object:	*my-order*		class(es): *transport-task*		
attribute(s):	name	range	min	max	filler(s)
	customer	*person*	1	1	—
	cargo	*good*	1	∞	—
	collect-from	*location*	1	1	—
	deliver-to	*location*	1	1	—
	distance	*integer*	1	1	—

Extending the class list consists in adding a class that is independent of those the object is an instance of. Again, the candidates are determined by KRIS, which

[2] For the sake of brevity we present abstractions of the actual windows. Figure 4 shows what they look like in reality.

proceeds in three steps. First it computes the set of possible concepts for the object at hand. Then, this set is filtered to yield those concepts that neither subsume nor are subsumed by a concept corresponding to a class in the current list. Finally, KRIS returns the most general concepts in the reduced set. Assume that in our example we want to extend the list of classes *my-order* belongs to. In this case TACOS offers only *single-task* and *recurring-task*, since *storing-task* is disjoint from *transport-task*, to which *my-order* belongs.

The attribute level. In the attribute level part the following items are displayed for each attribute of an object: the attribute's name, a list of classes restricting its range, the cardinality constraints, i.e., a lower and an upper bound for the number of fillers—indicated by the keywords min and max—and a list of the currently known fillers. Similar to the object level, one can refine a class in the range and extend the range by another class. Additionally, the user is allowed to strengthen the cardinality constraints and to add fillers.

Refining a class in the range of an attribute is similar to refining a class in the object level part. The possible refinements are computed by KRIS based on the test for possible range restrictions. For instance, suppose that in our example we have decided to refine the range of the attribute *collect-from* of *my-order*. Assume further that *location* has the two immediate subclasses *foreign* and *inland*, which are disjoint. Then both are suggested as possible refinements. For the sake of our example, suppose we choose *inland*.

Analogously, *extending* a list of range restrictions resembles extending the list of classes an object is member of. Again, the computation of the candidate classes suggested by TACOS involves the test for possible range restrictions.

Note that neither refining a class in the list of range restrictions nor extending the list results in displaying new attributes for the current object. However, as will be discussed below, it may result in additional attributes of the fillers when the consequences of these operations are computed.

The next operation we want to illustrate is the *introduction of attribute fillers*. As filler of an attribute, the user can either create a new object, i.e., one that does not already appear in the KB, or he can take an existing object.

In the second case, he can choose between (1) possible fillers, which comprise all objects that can be added as fillers without turning the static KB into an inconsistent state, and (2) plausible fillers, which are the subset of possible fillers already satisfying the constraints imposed on arbitrary fillers of the attribute. They are computed by means of the tests for possible and plausible fillers in KRIS. Coming back to the example, recall that in our KB *Berlin* and *Bonn* are instances of *inland*, and *London* is an instance of *foreign*. Then the only possible (and plausible) fillers of *collect-from* for *my-order* are *Berlin* and *Bonn*, since we already have refined the range of *collect-from* to *inland*. For the attribute *deliver-to*, also *London* is plausible. If we choose *Bonn* and *Berlin*, the following information appears:

object:	*my-order*		class(es): *transport-task*		
attribute(s):	name	range	min	max	filler(s)
	customer	*person*	1	1	—
	cargo	*good*	1	∞	—
	collect-from	*inland*	1	1	*Bonn*
	deliver-to	*location*	1	1	*Berlin*
	distance	*integer*	1	1	—

In the abouve example, all possible fillers are also plausible. In order to see the difference between the two, assume that there is also an instance *joe* of the class *person* in the KB, and assume further that *person* and *location* are not disjoint. Then also *joe* would be a possible filler, but not a plausible filler.

Finally, the user can *strengthen the cardinality constraints* imposed on the number of fillers of a given attribute. More precisely, he can increase the lower bound and decrease the upper bound. Again the possible values are offered in a mouse-sensitive menu. In our example, we can change both min and max of *cargo*. Since min and max coincide for any of the remaining attributes, they cannot be changed without running into an inconsistent state. When we attempt to modify them, TACOS will inform us about this fact. Moreover, it will not accept additional fillers for *collect-from* and *deliver-to*, because the cardinality constraints require exactly one.

5.2 The reasoning process

During the acquisition process, continuously new information entered by the user is passed to the domain modeling component. However, the object windows do not immediately display inferred facts. In order to view such derived information, the user has to start an update process. The process consists of two interleaving phases: realization of objects and application of rules. When the update has been completed successfully, the new state of the KB is pushed on a stack containing the states reached so far.

In the *realization phase*, all instance relationships between the objects and the classes occurring in the KB are computed. As a consequence, the system presents the most specific information about each object in the windows. In our example, TACOS recognizes that *my-order* not only belongs to *transport-task*, but also to *domestic-transport-task*. In fact, since the only filler of *deliver-to*, namely *Berlin*, is an instance of *inland*, the system concludes that every filler of *deliver-to* belongs to *inland*; in other words, the range of *deliver-to* is specialized to *inland*.

The *application of rules*, as described in Section 3 may cause functions to be called and new assertions to be added. Any newly added assertions, on the other hand, require a further classification phase. In our example, the rules in Figure 3 are applicable. As *my-order* is an instance of *transport-task* (although just classified as *domestic-transport-task*) and has fillers of *collect-from* and *deliver-to*, the value

computed by *compute-distance(Bonn,Berlin)*, say *598*, is bound to the variable d and the rule fires. Hence the assertion $(\textit{my-order}, 598) : \textit{distance}$ is added to the KB. Now the preconditions of the second rule are fulfilled and the function call in its consequence part is executed. After the update the object window of *my-task* looks as follows.

object:	*my-order*		class(es):		*domestic-transport-task*
attribute(s):	name	range	min	max	filler(s)
	customer	*person*	1	1	—
	cargo	*good*	1	∞	—
	collect-from	*inland*	1	1	*Bonn*
	deliver-to	*inland*	1	1	*Berlin*
	distance	*integer*	1	1	*598*

In a similar way as discussed so far, the user will enter during the next steps his personal data and a description of the cargo. If the task complies with the conditions for acceptance, as specified by rules, they can be forwarded to a data base that keeps track of accepted orders.

5.3 Handling inconsistencies

When TACOS is started with an initial KB it computes its consequences by repeated classification and rule application. After these operations it should arrive at a consistent state, since otherwise no meaningful work can be done.

From this moment on, the user is allowed to enter new assertions. As pointed out before, he cannot input arbitrary facts, but must choose from lists of possible items presented by the system. The underlying DL inference component guarantees that adding any of the items offered keeps the static KB consistent. However, this is no more true when also the *dynamic* part is taken into account, because arbitrary assertions can be added through the application of rules.

Therefore, after each reasoning step (see Section 5.2) the inference component checks whether the static KB is consistent. When an inconsistency is detected, the user is informed and the system backtracks to the previous state.

6 Discussion

We have experimented using TACOS with different domain models. One of them describes the possible tasks that can be executed by a shipping company and has served as illustration in this paper.

As another experiment we have translated the Wines knowledge base, originally formulated in CLASSIC [3], into TACOS. The Wines KB contains descriptions of meals and wines together with rules that impose constraints on the wines to accompany

the meal. Loaded into TACOS, a user can specify his meal while the system gives him advise on the beverage. In this scenario, two services can well be demonstrated: on the one hand, through the various refinement operation, one can stepwise traverse the space of possible meals, on the other hand, constraints on wines are accumulated and combined into a query, which then is used to find a particular wine object that satisfies them.

Although a toy example, the Wines scenario is prototypical for a whole class of applications, whose common characteristic is the interactive configuration of a complex object, and where a tool like TACOS can give support. Such an object can be a meal or a transportation task, but it can equally well be a computer or some other complex technical system to be configured.

We also conceive of other areas where the ideas realized in TACOS could be applicable. As schemas of object-oriented data bases tend to be rich in semantic information, there is again the problem for the casual user of getting acquainted with a domain model, if he wants to formulate sensible queries. In such a situation, a system like TACOS could be employed to support a user in specifying not his inputs, but the objects he wants to retrieve.

We assume that the schema and its integrity constraints can be mapped into a TACOS domain model. Considering the entities described by a set of object windows as variables to be instantiated, one could incrementally formulate a query asking for families of objects. The power of the language would at least be sufficient to formulate conjunctive queries over unary and binary predicates (classes and attributes). Due to the assistance through a reasoner, it is impossible to come up with an inconsistent query, that is, a query that cannot have instances because of the integrity constraints of the database.

7 Conclusion

We have presented TACOS, an information acquisition interface, which receives its power from the inferential capabilities of the DL system KRIS.

Yolanda Gil has formulated a set of demands on knowledge acquisition tools [7]: (1) maximum guidance of the user, (2) robustness, in the sense that erroneous input is tolerated, or better, prevented, and (3) flexibility, in the sense that the system is applicable to a wide range of domains. We think that these requirements give also a good guideline for the development of task acquisition systems.

The TACOS system meets each of them in a specific way: It guides the user by displaying input patterns and actively suggesting items for input. It does not allow information to be entered that is meaningless with respect to the domain model, so that the need for error recovery is reduced. Finally, since its functionality is parameterized by a largely declarative domain model, it can be easily adapted to different applications.

References

[1] F. Baader and B. Hollunder. $\mathcal{KRIS}$: $\mathcal{K}$nowledge $\mathcal{R}$epresentation and $\mathcal{I}$nference $\mathcal{S}$ystem. *SIGART Bulletin*, 2(3), 1991.

[2] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithms. In M. Richter and H. Boley, editors, *Proc. of PDK'91*, Kaiserslautern, Germany, 1991.

[3] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, L. A. Resnick, and A. Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In J. Sowa, editor, *Principles of Semantic Networks*. Morgan Kaufmann, San Mateo, Calif., 1991.

[4] M. Buchheit, F. M. Donini, W. Nutt, and A. Schaerf. Refining the structure of terminological systems: Terminology = Schema + Views. In *Proc. of AAAI'94*, Seattle (Washington, USA), 1994.

[5] F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. Adding epistemic operators to concept languages. In *Proc. of KR'92*, Cambridge, Mass., 1992.

[6] B. Gaines and M. Shaw. Knowledge acquisition tools based on personal construct psychology. *The Knowledge Engineering Review*, 8(1), 1993.

[7] Y. Gil. Knowledge refinement in a reflective architecture. In *Proc. of AAAI'94*, Seattle (Washington, USA), 1994.

[8] P. Hanschke and K. Hinkelmann. Combining terminological and rule-based reasoning for abstraction processes. In *Proc. of GWAI'92*, Bonn, Germany, 1992.

[9] R. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3), 1991.

[10] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1990.

[11] P. F. Patel-Schneider, D.L. McGuinness, R. J. Brachman, L. Alperin Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bulletin*, 2(3), 1991.

Appendix

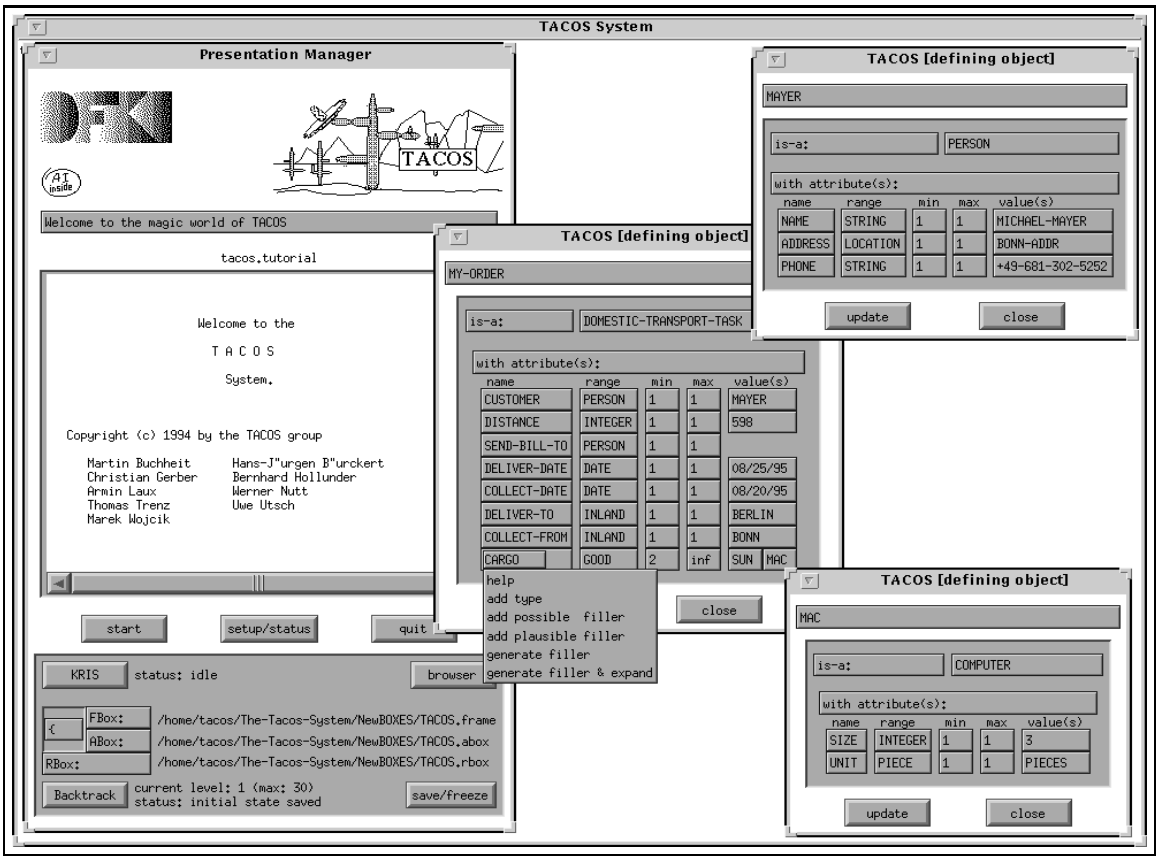

Figure 4: The TACOS user interface.