A Browser for Software Reuse

Panos Constantopoulos and Elena Pataki

Institute of Computer Science
Foundation of Research and Technology - Hellas
Heraklion, Crete, Greece
e-mail: {panos|pataki}@csi.forth.gr

Abstract

One important aspect of software reuse is the organization of collections of reusable software artifacts. The Software Information Base (SIB), developed within the ESPRIT project ITHACA, provides a directory to reusable software by storing information about software objects concerning the entire software life-cycle, namely requirements, design and implementation descriptions, as well as aggregate representations of complete systems and application domains. The SIB has an attributed graph structure. The selection of artifacts from the SIB, either directly or through other software development tools, is performed using a specialized Selection Tool (ST). In this paper we present the design and functionality of the Selection Tool. The main search mode supported by the Selection Tool is browsing. It is a flexible navigation process that takes full advantage of the knowledge representation mechanisms underlying the SIB semantic network, and provides local search of controllable size, direct access to specific areas or objects in the SIB, filtering mechanisms, and orientation aids. The information stored in the SIB and displayed by the Selection Tool is multimedia. The representational issues addressed by the SIB - ST system, as well as the relationship between the ST and hypertext systems are discussed.

1 Introduction

Software productivity is far from being considered satisfactory and several alternatives for improving it are being explored [1, 5, 21]. Reusing software components is a natural and very appealing idea, since the reuse of artifacts is encountered in all technical endeavours. Object-oriented programming, software libraries, AI-based design methods and organizational support are but some approaches to software reuse. On the other hand, it is commonly accepted that software reuse concerns not only code but the entire software development process, including requirements analysis, design, implementation and the development experiences gained in all stages. Thus source code is only one part of a reusable software object.

This paper presents the tool used for selecting software objects in connection with a software information base. The Software Information Base (SIB) is intended to store information about requirements, designs and implementations of software and supports selection of software through an associated Selection Tool (ST).

The whole system (SIB-ST) has been developed within the context of the ITHACA project, a large software engineering project sponsored by the European Communities through the ESPRIT programme, which aims at developing an integrated application

development environment based on object-oriented techniques. The ITHACA environment includes an object-oriented programming language and database, application development and support tools, and an evolving software base [8, 9, 33].

As the name indicates, the Software Information Base (SIB) stores *information about* software components, not the components themselves. It is meant to serve as a directory to a collection of software components in order to facilitate their reuse. Information about the early stages of software construction (requirements analysis, design) is important because any necessary modification of an object is easier to identify at those stages. The representation language used in the SIB is Telos [22]. This is an E-R based language specifically designed for the development of information systems. The preference for Telos over other extended E-R models is due to its treatment of metaclasses and attributes. The organization of the SIB is based not only on the usual conceptual modeling principles of classification, generalization and aggregation, but also on principles addressing particular user and methodological needs, such as modularization, versioning, semantic similarity and others. Besides, the contents of the SIB actually are multimedia objects created and used by different development tools which require them to be presented in corresponding particular forms (e.g., E-R diagrams, data flow diagrams).

The Selection Tool (ST) is the main communication point between the SIB and the external world, i.e. users and other software development tools (e.g. the ITHACA Visual Scripting Tool [23]. Its task is to extract information from the SIB and to provide an interface for presenting this information in an appropriate way. The ST views the SIB as an abstract data type through the SIB Query Processor. The latter filters the SIB and passes the information to the ST for further processing.

The ST has two major functional parts. A querying mechanism filters the SIB to produce a subset of it possibly containing candidates for reuse. A browser allows viewing parts of the SIB at various levels of detail and navigating in a hypertext-like fashion through them. The filtering and browsing functions are actually interleaved in the operation of the ST.

Choosing a software component for reuse from the SIB is an ill-structured decision problem and there is no "best" solution to it that an automated procedure could find. There are subjective factors in judging the suitability of a software component for reuse, therefore the user will eventually have to examine a set of candidates in order to select the one that best fits his needs.

Browsing suggests itself as a natural mechanism not only for examining a query answer set containing potential candidates for reuse, but also for exploring a subset of the SIB in small steps. Indeed, browsing is a navigational search process that exploits the references established between objects by the SIB structuring principles. The structure of the SIB is designed to be rich enough to ensure the effectiveness of browsing which, in view of the uncertain nature of the software selection process, is the search mode of choice. As we shall see, the browser of the SIB has a substantial resemblance to a hypertext system. Consequently, the ST has the functional advantages of hypertext systems yet it also has to deal with their problems.

In section 2 we introduce the contents and structure of the SIB and the selection mechanisms. In section 3 we describe the functionality and current implementation of the

Selection Tool. In section 4 we discuss the relationship of the ST with hypertext systems and planned further development.

2 Description and Selection of Reusable Software

The SIB contains information about software objects in the form of a variety of descriptions of software objects and semantic relations that hold among them. Moreover, aggregations of descriptions pertaining to specific systems or even to entire application domains are defined as Application Frames. The notion of the Application Frame (AF) is central to the scenario for software reuse adopted in ITHACA [8, 33].

2.1 Structure of the SIB

The exposition in this subsection mostly follows [10]. The SIB is structured as an attributed directed graph the nodes and links of which represent *descriptions* of software objects and semantic relations respectively.

There are three kinds of descriptions:

- requirements descriptions (RD);
- design descriptions (DD); and
- implementation descriptions (ID).

These descriptions provide three corresponding views of a software object:

- an application view, according to a requirements specification model (e.g., SADT);
- a system view, according to a design specification model (e.g., data flow diagram); and
- an implementation view, according to an implementation model (e.g., set of C++ classes together with documentation).

Descriptions can be simple or composite, consisting of other descriptions. The term *descriptions* reflects the fact that these entities only describe software objects. The objects themselves reside outside the SIB. Descriptions are related to each other through a number of semantic relations listed below. In addition to the usual isA, instanceOf and attribute relations supported by object-oriented data models and knowledge representation schemes, several special attribute categories have been defined for the purposes of the SIB. The SIB is defined, as mentioned, in terms of the Telos knowledge representation language which supports creating an infinite instantiation hierarchy and treats attributes as objects in their own right (which, therefore, can also have attributes). These features of Telos are fully exploited in structuring the SIB.
The following relations are supported in the SIB (the link names are those appearing on the ST interface, see fig. 1):

(1) *Attribution*, represented by *attribute* links. This is a general, rather unconstrained representation of semantic relations, whereby the attributes of a description are defined to be instances of other descriptions. An attribute can have zero or more values.
Example:

```
Description SoftwareObject with
    attributes
```

```
        author: Person
        version: VersionNumber
```

Software Object has attributes author and version whose values are instances of Person and VersionNumber respectively.

(2) *Aggregation*, represented by *hasPart* links. This relates an object to its components which have to be objects of the same kind.
Example:

```
Description SoftwareObject with
   ...
   hasPart
       components: SoftwareObject
```

The components of an object have a distinct role in the function of the object and any possible changes to them affect the aggregate object as well (e.g., new version).

(3) *Classification*, represented by *instanceOf* links. Objects sharing common properties can be grouped into classes. An object can belong to more than one classes. Classes themselves are treated as generic objects which their members are instances of and which, in turn, can be instances of other, more generic objects. In fact, every SIB object has to be declared as an instance of at least one class. Thus, an infinite classification hierarchy is established starting with objects that have no instances of their own, called tokens. Multiple instantiation is allowed. Instantiation of a class involves instantiating all the associated semantic relations. Thus relations are treated as objects themselves.
Example:

```
Description BankIS instanceOf SoftwareObject with
    author
         : Panos
    version
         : 0.1
    components
         : CustomerAccounts, Credit, Investments
```

The attribute and hasPart links of BankIS are instances of the corresponding attribute and components links of SoftwareObject.

Classification is perhaps the most important modeling mechanism in the SIB ([34], and [25] give a detailed account of the construction of models and descriptions in the SIB).

(4) *Generalization*, represented by *isA* links. This allows multiple, strict inheritance of properties between classes leading to the creation of multiple generalization hierarchies. A class inherits all the attributes of its superclasses (one or more, multiple inheritance), however inherited properties can only be constrained, not overridden (strict inheritance).

(5) *Correspondence*, represented by *correspondsTo* links. A software object can have zero or more associated requirements, design and implementation descriptions. In fact, a

requirements specification may generate more than one alternative designs and a design may give rise to more than one implementations. Correspondence links denote such correspondences between requirements, design and implementation descriptions of a single software object.

Correspondence links define the internal structure of application frames. An *Application Frame* (AF) is a construct of coarse granularity in the SIB which represents a complete system or family of systems and comprises (*hasPart*) at least one implementation and optional design and requirements descriptions. Thus AF's encapsulate all the information pertaining to specific applications regardless of complexity (e.g., inventory monitoring or complete production planning and control system). In addition they support a natural organization of the SIB by application domain.

(6) *Similarity*, represented by *similarTo* links. The similarity relation is defined between objects of the same kind (i.e., RD, DD, ID, or AF) and has two attributes: a similarity criterion and a similarity measure. Two objects are said to be similar with respect to some criterion if they can substitute one another with regard to this criterion. The similarity criterion can be endogenous, defined in terms of relations already stored in the SIB, or exogenous, provided by the user who specifies a particular similarity link. Accordingly, similarity links can be computed or user-defined. The similarity measure expresses the degree to which the substitution of two similar objects, in either direction, is satisfactory and is a number in the range [0,1]. The similarity criterion is a mandatory attribute while the measure is optional.

Similarity links give rise to equivalence classes, possibly endowed with their own internal distance measures, which can support the application of analogical reasoning in software reuse and development.

(7) *Specificity*, represented by *specialCaseOf* links. This relation is defined only between application frames to denote that one application frame is less parameterized than another. E.g., a bank accounting and a hotel accounting application frame could both be derived from a more general, parametric accounting application frame.

2.2 Selection in the SIB

The SIB system offers a number of maintenance, selection and workspace management functions. Maintenance functions include insertion, deletion and update of information in the SIB and are supported by appropriate textual and graphical editors. Selection functions include querying and browsing the SIB for purposes of selecting reusable artifacts. And workspace management involves the dynamic definition and modification of workspaces to provide easier and more efficient interaction with the SIB [8].

The selection of software descriptions from the SIB is accomplished through the *Selection Tool* (ST) and it is an iterative process comprising alternate stages of retrieval and browsing. Browsing usually is the final and often the only stage of the process. The functional difference between the retrieval and the browsing mode is that the former supports the retrieval of an arbitrary subset of the SIB while the latter supports local exploratory searches withing a given subset of the SIB. Operationally, both selection modes address queries to the SIB.

Seen as operations on an abstract data type, the selection functions can be defined as follows [10]:

Retrieve: Query x Associations → SetOf (Descriptions , Weights)

Browse: Identifier x Links x Associations → Views

Associations can be seen as groupings of descriptions (see [10] for details). The SIB system is intended to support non-Boolean queries. Query conditions consist of logical combinations of predicates which are, in general, fuzzy. A real number between 0 and 1, called *weight* is associated with each predicate. Boolean predicates are simply special cases with weight 1 if the value of the predicate is 'yes' and 0 if the value is 'no'. The result of a query is a set ranked by weight. Weights of logical expressions are computed on the basis of the following rules:

Let *p* and *q* be predicates with weights *w(p)* and *w(q)* respectively. Then

w(p OR q) = max {w(p), w(q)}

w(p AND q) = w(p) * w(q)

w(NOT p) = 1 - w(p)

Alternative rules for the computation of weights exist (e.g., see [16, 29]), as well as alternative retrieval models altogether. An important feature of the present approach (though not unique to it) is the ability to express such notions as similarity or affinity [26] in terms of weights on fuzzy relations which give rise to fuzzy predicates. Boolean queries are merely a special case.

The *Retrieve* function takes as input an association and a (compound) non-Boolean query and returns a subset of the association with weights attached indicating the degree to which the descriptions in the answer set match the query. In the current implementation only Boolean queries are supported, yet the extension to non-Boolean is currently undertaken.

Browsing clearly is a special retrieval operation. It starts at a specified SIB description which is the focus of attention and is called *current object* and produces a view of a neighbourhood of interest of the current object within a given association. Since the SIB has a network structure, the neighbourhood of the current object (node) is defined in terms of the links of interest adjacent to it. As we shall later see, the size of the neighbourhood can also be controlled. Thus, the *Browse* function takes as input the identifier (name) of the current object, a list of names of link classes of interest and an association, and determines a local view centered around the current object. By calling *Browse* again with the identifier argument equal to the name of one of the objects contained in the browser's view, that object is made current and the view is updated. Effectively, the *Browse* function provides a moving window with controllable filters and size, which allows navigational search over subsets of the SIB network. The default association is the entire SIB.

The multimedia nature of SIB descriptions calls for the development of a hypermedia annotation mechanism that would gracefully complement the SIB semantic network. This is accomplished by establishing referential links between descriptions, treated as a special category of attribute links, thus completely integrated in the SIB network model. Hypermedia annotations include text, graphics, raster images and algorithm animations.

3 The Selection Tool of the SIB (ST)

Queries to the SIB can be classified as *explicit* or *implicit* and as *interactive* or *programmatic*. An *explicit* query involves an arbitrary predicate explicitly formulated in a query language or through an appropriate form interface. An *implicit* query, on the other hand, is one generated as a result of navigational commands in the browsing mode, or one of particular significance and frequent occurrence, "pre-canned" for ease of use and offered as a button or menu option. An *interactive* query is formulated through the user interface of the ST, while a *programmatic* one is submitted by another software development tool, such as the Visual Scripting Tool of ITHACA, through its own user interface.

The ST is designed to support all the relevant kinds of queries. In the current implementation, priority has been given to *implicit, interactive* queries. As already explained, these correspond to the browsing selection mode and are expected to take up the majority of the interaction with the ST. A form-based query interface, closely related to the data entry form of the SIB, as well as a query language are being developed to support *explicit, interactive* queries. Finally, *explicit, programmatic* queries are supported by the ST providing a programmatic interface to other tools, through which the primitive query operations of the Telos system underlying the SIB can be used.

In what follows we present the browsing functions of the ST.

3.1 Functionality of the Selection Tool

The ST user interface consists of the following windows (fig.1):

- a *Graphical Browser* which displays a part of the network of the SIB around a selected object (current object).
- a *Link Filter* with buttons, each corresponding to a link type, used for filtering the information displayed by the Graphical Browser.
- a *History List* which keeps a record of users' moves through the SIB network.
- an *Application Frames List* listing the contents of the entire SIB in terms of application frames.
- a *Main Form* which shows information about the current object.
- an *Auxiliary Form* which displays information about any other object, after a selection made by the user in a hypertext-like manner on the Main Form.
- a *Function Menu* which offers a variety of useful functions.

In what follows we describe the functionality of each component.

- *Graphical Browser*:

The Graphical Browser, built using the LABY graphical editor, displays a part of the SIB network around a selected object (*current object*). The structures currently displayed have the form of a star whose central node is the current object. The window of the Graphical Browser is semantically divided in two parts. From each node appearing in the lower part emanates at least one link pointing to the current node. Similarly, there is at least one link emanating from the current node and pointing to each node appearing in the

upper part. The types of links are denoted by a colour code. For example the isA relation is shown with red links while the instanceOf relation with green links. The colour code is shown in the Link Filter. Nodes appearing in the graph of the browser are selectable with the mouse. When selected, a node becomes current, it is placed in the middle of the display and a move in the SIB network results. The links displayed include direct links from the current object to other objects and computed isA and instanceOf links.

The population of the display is controlled by means of the *Link Filter* (see below) and the *Instance Box*. The Instance Box appears in the display of the Graphical Browser when the instances of a certain active link class (i.e. selected by the Link Filter) adjacent to the current object are too many to be shown on the display. On selecting the Instance Box of a link class with the mouse, a list of objects related to the current one by that type of links appears (see fig.4). The objects on this list are selectable just as those displayed graphically.

- *Link Filter*:

The Link Filter provides buttons corresponding to link classes and is used for activating/deactivating links thus controlling the information displayed in the Graphical Browser. Each button acts as a filter on a certain relation. If the button is selected (on), the corresponding relation is displayed and vice versa. The isA and instanceOf buttons further offer the option of displaying computed in addition to direct isA and instanceOf links. All buttons show the colour code of the link classes and have a help facility. When moving from node to node in the Graphical Browser, the state of the Link Filter does not change unless the user does so explicitly.

- *History List*:

The History List is a navigation aid intended to prevent users from getting lost, a common problem in hypertext systems [6]. The History List is scrollable and contains the names of the objects selected as current during a session in chronological order, the most recent one shown at the bottom (as in the history command of Unix). All entries of the list are selectable. A selection made on the History List is functionally equivalent to one made on the Graphical Browser.

- *Application Frames List*:

The Application Frames List contains the names of the application frames, reflecting the overall structure and contents of the SIB. The purpose of the Application Frames List is to compensate for the limited scope of the Graphical Browser, which is a shortcoming if an extended area of interest is sought by navigation rather than by explicit querying. The application frames are displayed in an indented list representing the existing hierarchical structure. Each item on the list is selectable, which effectively allows big steps over the SIB network in the browsing mode.

Moreover, the Application Frames List serves as the initial entry point to the ST.

- *Main Form*:

The Main Form is the top right-most window of the ST and invariably displays information about the current object. It shows the abstracted definition of the object in a form layout. The information presented is generated by unparsing the SIB. The form can

also contain multimedia annotations to the object, each one displayed in a separate window. At present, this annotation is textual and graphical (fig.5), but can be of any other type with no additional effort, provided that appropriate tools exist in the working environment.

- *Auxiliary Form*:

To get information about a displayed object, other than the current one, without changing the actual view in the Graphical Browser, the user can select the name of the desired object on the Main Form in a hypertext-like fashion (see fig.1). An Auxiliary Form will appear at the bottom right of the ST and display information about the selected object in the manner of the Main Form. To prevent user distraction, only one auxiliary form can be open at a time; also objects are not selectable on auxiliary forms. The auxiliary form is actually a preview mechanism and is offered as an orientation aid.

- *Function Menu*:

Through the Function Menu, several other windows for performing various useful functions can appear on demand. Currently these functions are:

- *GotoObject*: Allows direct access to invisible objects by name.
- *EnterData*: A Data Entry Form is offered for entering data into the SIB (fig. 6). The Query Form, currently under construction, will have a similar appearance.
- *KeepObject*: Keeps a retrieved object in a local workspace.
- *Iconify* and *Quit*: Closes and iconifies a window; and quits the ST respectively.

3.2 Usage Example

Suppose we would like to add in the SIB an application dealing with processing of letters, which will assist secretaries, managers, and others to write professional letters, check them, and, after final approval, mail them through post or electronic mail. Looking at the Application Frames List, we observe that there is already an Office Information System called `WooRKS`. The basic concepts in `WooRKS` are actors, roles and procedures.

Our starting point will be `WooRKS`, which we select through the Application Frames List. As we can read in the natural language comment attached to the corresponding Main Form, `WooRKS` is a work flow system for offices, which handles a variety of activities (see fig.1). So this might be one of the candidate places to search for a letter processing application. `WooRKS` has three attributes which are further explained in its Main Form. One of them, *reqDescr*, deals with `WooRKS` requirements descriptions and it will probably provide us with more information about what the `WooRKS` system actually does. Before making it current, we preview its contents on the Auxiliary Form, by selecting `WooRKS1_RD_FORM` from the Main Form, and we decide to visit it. In figure 2 `WooRKS1_RD_FORM` is current in the Graphical Browser window. We observe that it handles the following *classes* of activities: `OrderProcessing`, `WarehouseProcessing`, and `AccountProcessing`. `OrderProcessing` sounds the most close to letter processing, since letter and order processing share some operations, such as checking and archiving. We decide to visit `OrderProcessing` to see if it includes

what is needed for letter processing in general. After examination we conclude that this is not true. We return to WooRKS1_RD_FORM through the History List and decide to preview WarehouseProcessing and AccountProcessing to see if there is anything relevant there. By previewing these nodes we realize that none of them fulfills our needs. We further notice that they are all instances of FormProcessClass.

We take a closer look on FormProcessClass by moving to it through the GotoObject facility (fig.3). As FormProcessClass is a subclass of FormClass, it inherits its attributes. By previewing FormClass, we find out that it has two attributes, *roles* and *baseRole* (see fig.3) and decide to create a new instance of FormProcessClass, called LetterProcessing, whose *roles* will correspond to the initial requirements imposed on our letter processing application. In particular, the *baseRole* of LetterProcessing will be LP_base_role, and the *roles* will be LP_letterCompose, LP_letterCheck, LP_letterApprove, LP_letterSend, LP_letterReceive, and LP_letterArchive.

We have chosen this convention for naming the *roles* by analogy to the existing *roles* of the other activities. To see these names we first made FormRole current using the GotoObject facility (see fig.4). Since FormRole has too many instances to be displayed on the Graphical Browser, an Instance Box appears by clicking on the "MANY INSTANCES" box of the Browser.

At this point we start creating the *roles* and *baseRole* of LetterProcessing. Before creating LP_letterArchive, we visit OP_orderArchive by selecting it from the Instance Box (fig.4). This act is worthwhile because we find a *correspondsTo* link from OP_orderArchive to ArchiveAct, which is an instance of ADMActivity (see the Main Form in fig.5). Knowing that ADMActivity handles the design descriptions, we can further proceed by defining the ADMActivity corresponding to the new LP_letterArchive in a similar way, or even use the ArchiveAct as the ADMActivity of LP_letterArchive. Similarly, we may use CompileRefAct and/or EvaluationOrderAct which *correspondTo* OP_orderCheck as the ADMActivity of LP_letterCheck, etc.

Finally, we are in a position to define the new LetterProcessing node. We move to FormProcessClass using the GotoObject option and use the EnterData facility, which will make our task easy, even if we have no knowledge of the syntax of Telos. The Data Entry Form for LetterProcessing is shown in figure 6. In the same figure you can see the results of entering the information.

3.3 Implementation

The SIB system consists of the following major parts (Figure 7):

- The *SIB Interactive User Interface* generates and coordinates the other parts, including the interface tools of the Data Entry Forms and the Selection Tool, except for the Graphical Browser. It is implemented using the OSF/Motif toolkit.

- The *Graphical Browser* presents parts of the SIB network graphically and allows the user to browse through it by sending messages to the SIB Interactive User Interface in response to user actions. The Graphical Browser is a LABY graphical editor with only the working area present.

- The *Display Forms* are used to provide information about the current node or another selected node in a form layout. They are designed to support multimedia information (text, graphics, images, animation, etc.).
- A special *Data Entry Form* provides for entering data into the SIB through a form interface.
- The *Query Interface* handles queries about SIB objects, issued to it by the various components of the ST or by the Data Entry Form. As a result of processing a query it constructs a file suitable for display by the Graphical Browser or the Data Entry Form. In the current implementation a separate query interface (based on the client-server model) is used for programmatic queries. However, the two query interfaces will be integrated in the future.

LABY is a general purpose graphical editor developed in part within the ITHACA project [19]. The entire SIB-ST prototype system has been implemented in C++, including the underlying Telos system. The Telos implementation has particularly emphasized query performance and efficiency of memory usage (for more details see [8]). The system runs on Sun3, Sun4 series, SparcStations and 386 machines under Unix and requires the X window system and a colour monitor.

4 Discussion

[20] identifies the following three steps in reusing software: selection, specialization, and integration. Abstraction plays a central role in each of these steps. It makes artifacts easier to understand and to specialize, by chosing specific realizations, and allows information about their interface to be abstracted from the definitions. In the work reported here we are concerned with abstractions for supporting the process of selecting software for reuse.

The same steps of the reuse process are identified by Prieto-Diaz [27, 28], with an additional evaluation step, when there is no exact match with the imposed requirements. The candidates for reuse are ranked according to the adaptation and conversion effort they require before reuse can actually take place. In this approach, a faceted classification scheme supports both the retrieval process and the evaluation and adaptation of code segments. The terms in each facet are organized in a directed acyclic graph with weights indicating their conceptual closeness. The selection of the facets and terms (namely the faceted classification schedule) is a process that requires careful consideration in order for the classification scheme to be successful. The system allows for a flexible change of the classification schedule as implied by the specific application domain the library is supposed to cover. The query process allows the modification, generalization/specialization and expansion of queries while vocabulary control is assisted by a dictionary of synonyms. Despite the shortcomings of defining the conceptual distances, the classification scheme is promising, yet the query interface is suitable for expert users only. Browsing through the conceptual graph would be an easier and, in some cases, more efficient retrieval method.

User interface and functionality issues have, in fact, been so far one of the chief three problems that have hindered the acceptance of database technology in software engineering, the other two being the technical support (efficiency, safety) and the

representational power of data models. Early software databases provided rather primitive interfaces for selection (declarative or navigational) and no explicit support for data entry. Even elementary configuration management features, such as the re-configuration of objects as a result of component changes, are hardly supported by databases (an exception is CACTIS [18]).

[2] recognize the importance of good documentation for understanding the reusable components. Hypertext systems allow the attachment of various annotation elements interconnected with each other, to components, and provide instant access to this supporting information. The SIB allows the user to relate any kind of annotation to a description. Furthermore, our plans for a Hypermedia Annotation Mechanism, explained below, will greatly support the understanding of the reusable components.

The node and link model of the SIB resembles the hypertext model. In both, nodes and links are of equal importance, serving as units of information and as orientation cues. The type of link emanating from a given node conveys information about the target node. In addition, the name of a node usually reflects its content. This helps the user find a path to his destination. The question naturally arises of whether an existing hypertext system could provide a satisfactory interface to the SIB. It turns out that the full representational power of Telos, underlying the SIB, cannot be rendered by one of the currently available hypertext systems known to us, let alone efficiency issues arising from heterogeneity. The tools that compose the Selection Tool (e.g. the Graphical Browser and the Link filter, the Application Frames List, the Forms, etc.) take advantage of the underlying knowledge base in order to help the user locate and reuse the software modules. Moreover, given that the SIB will be highly populated, other features common in most hypertext systems (e.g. global browsers) or the ability to directly following links are not enough to prevent *disorientation* and *cognitive overhead* [6], which are inherent problems of hypertext systems. On the other hand, using Telos, we provide users with pre-defined views (e.g. Application Frames List), constructed by querying the SIB and updated dynamically reflecting changes in it. Such views reduce the cognitive overhead problem. The History List combined with the ability to preview the contents of a specific object on the Auxiliary Form before actually visiting it prevent disorientation.

Of a number of advanced hypertext systems, such as gIBIS [7], DIF [13, 14], Intermedia [15, 35], NoteCards [17, 32] and Neptune [3, 12], all of which have some of the desirable features for a selection tool of the SIB, DIF is designed to serve a purpose similar to the SIB-ST system, namely to support a software life-cycle with emphasis on reuse. However, the basic reusable constructs of DIF, *Basic Templates* and *Standardized forms*, have only one level of instantiation and the system relies upon the use of keywords to preserve the consistency of information. Structural information is kept in an Ingres database, which has to be queried or navigated through to locate reusable Basic Templates. Unfortunately, the searching process cannot use the actual information stored using software engineering tools for functional and architectural specifications and this limits the flexibility of the selection process.

The current version of the SIB prototype system, including the ST presented here, has been subjected to relatively extensive experimentation by users inside and outside the implementors' organization, indicating a high acceptance level. Nevertheless, no formal evaluation experiments have yet been conducted. A number of desirable improvements,

on the other hand, have been identified [24], briefly listed below.

Several enhancements to the Graphical Browser are planned. First, the presentation of all direct links on the display, not only the ones adjacent to the current node, so as to show all the direct relations that exist between nodes in the local view. Second, the option to control the size of the displayed neighbourhood of the current node will be provided. In a graph-theoretic sense, the size of the neighbourhood currently is 1 : each path connecting the current node to another node in the neighbourhood contains exactly one link. This restricts the view to a small area and imposes a small step size during navigation. On the other hand, it also keeps the information in the browser's window from exploding. Control over the neighbourhood size will be granted by attaching *scope parameters* to the buttons of the Link Filter. The value of a scope parameter will determine the size of the neighbourhood with regard to the corresponding type of links. Third, the layout of the browser's display will be re-arranged so that the spatial distribution of links will be directed by their semantics.

The History mechanism will be enhanced to support the reuse of paths [36] once the *context* mechanism of Telos [8] is available.

Similarly, the context mechanism will enable the definition of context-specific Application Frames Lists. Highlighting the current view area in the Application Frames List is an obvious further orientation aid.

Given that *attribute* links can be specialized to represent user-defined relations, the question naturally arises of representing such relations in the Link Filter. A user-configurable Link Filter with the present one as a default is a possible solution. Furthermore, it has been noted that a sequence of changes of state in the Link Filter results in an unpleasant sequence of updates of the display. This can be avoided by committing all the changes at once at the user's command.

The free-text comments that appear in the Forms serve as a primitive annotation mechanism. We are currently exploring ways to provide an enhanced Hypermedia Annotation Mechanism by either incorporating in the SIB one of the existing hypertext models [4, 11, 30, 31] or developing a new one that will best serve our needs for flexibility and efficiency.

Finally, the Query Form, currently under development, will complete the Selection Tool. The Query Form will offer a form-based query interface similar to the Data Entry Form (fig. 4) and the option to formulate queries directly in a query language.

References

1. J. Bigelow, ''Hypertext and CASE,'' *IEEE Software*, March 1988.
2. Ted Biggerstaff and C. Richter, ''Reusability, Framework, Assessment & Directions,'' *IEEE Software*, vol. 4(2), March 1987.
3. Brad Campbell and Joseph M. Goodman, ''HAM: A general purpose Hypertext Abstract Machine,'' *Communications of the ACM*, vol. 31(7), pp. 856-861, July 1988.

4. R. Caudillo and M. Mainguenaud, ‘‘A Hypertext - Like Multimedia Document Data Model,’’ *Int’l Conf. on Multimedia Information Systems*, pp. 221-241, McGraw-Hill, 1991.

5. E. Chikofsky and Rubenstein B., ‘‘CASE: Reliability Engineering for Information Systems,’’ *IEEE Software*, March 1988.

6. J. Conklin, ‘‘Hypertext: An Introduction and Survey,’’ *IEEE Computer*, September 1987.

7. J. Conklin and M. Begeman, ‘‘gIBIS: A Hypertext Tool for Exploratory Policy Discussion,’’ *ACM Tr. on Office Information Systems*, vol. 6(4), October 1988.

8. P. Constantopoulos, M. Doerr, E. Pataki, E. Petra, G. Spanoudakis, and Y. Vassiliou, *The Software Information Base-Selection Tool integrated prototype*, Institute of Computer Science, Foundation of Research and Technology - Hellas, Heraklion, Crete, January 12 1991.

9. P. Constantopoulos, M. Jarke, J. Mylopoulos, B. Pernici, E. Petra, M. Theodoridou, and Y. Vassiliou, *The ITHACA Software Information Base : Requirements, Functions, and Structuring Concepts*, Institute of Computer Science, Foundation of Research and Technology - Hellas, Heraklio, Crete, May 1989.

10. P. Constantopoulos, M. Jarke, J. Mylopoulos, and Y. Vassiliou, *Software Information Base - A Server for Reuse*, Institute of Computer Science, Foundation of Research and Technology - Hellas, Heraklion, Crete, November 1991.

11. W.B. Croft and H. Turtle, ‘‘A Retrieval Model for Incorporating Hypertext Links,’’ *Hypertext ’89, Proc.*, pp. 213-224, Pittsburgh, Pennsylvania, November 5-8, 1989.

12. Norman M. Delisle and Mayer D. Schwartz, ‘‘Contexts - A Partitioning Concept for Hypertext,’’ *ACM Tr. Office Information Systems*, vol. 5(2), pp. 168-186, April 1987.

13. P.K. Garg and W. Scacchi, ‘‘Composition of Hypertext Nodes,’’ *Proceedings of the 12th Online Information Meeting*, London, December 6-8, 1988.

14. P.K. Garg and W. Scacchi, ‘‘A Hypertext System to Manage Software Life-Cycle Documents,’’ *IEEE Software*, vol. 7(3), pp. 90-98, May 1990.

15. Garrett, Smith, and N.K. Meyrowitz, ‘‘Intermedia: Issues, Strategies and Tactics in the Design of a Hypermedia Document System,’’ *Proc. Conf. on Computer-Supported Cooperative Work*, MCC Software Technical Program, Austin, Texas, 1986.

16. S. Gibbs, ‘‘Querying Large Class Collections,’’ *Object Management (D. Tsichritzis, ed.)*, Centre Universitaire d’ Informatique, Universite de Geneve, 1990.

17. F.G. Halasz, ‘‘Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems,’’ *Communications of the ACM*, vol. 31(7), pp. 836-852, July 1988.

18. S.E. Hudson and R. King, ‘‘Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System,’’ *ACM Trans. on Database Systems*, vol. 14(3), September 1989.

19. M. Katevenis, T. Sorilos, C. Georgis, and P. Kalogerakis, *LABY User's Manual*, Computer Science Institute, Foundation of Research and Technology, Heraklio, Crete, May 1990.

20. C.W. Krueger, *Models of Reuse in Software Engineering*, Carnegie Mellon, December 1989.

21. C. Martin, ''Second Generation Case Tools: A Challenge to Vendors,'' *IEEE Software*, March 1988.

22. J. Mylopoulos and others, *TELOS: Representing Knowledge about Information Systems*, Institute of Computer Science, Foundation of Research and Technology - Hellas, Heraklion, Crete, August 1990.

23. O. Nierstrasz and others, ''Objects + Scripts = Applications,'' *Esprit '91*, pp. 534-552, Commission of the European Communities, 1991.

24. E. Pataki and P. Constantopoulos, ''The Selection Tool of the Software Information Base: A Hypertext Perspective,'' *Working Paper*, Institute of Computer Science, Foundation of Research and Technology - Hellas, Heraklion, Crete, November 1991.

25. E. Petra and C.V. Vezerides, *SIB Content's Manual*, Institute of Computer Science, Foundation of Research and Technology - Hellas, Heraklion, Crete, January 1991.

26. X. Pintado, ''Selection and Exploration in an Object-Oriented Environment: The Affinity Browser,'' *Object Management (D. Tsichritzis, ed.)*, Centre Universitaire d' Informatique, Universite de Geneve, 1990.

27. Ruben Prieto-Diaz, ''Implementing Faceted Classification for Software Reuse,'' *Communications of the ACM*, vol. 34(5), pp. 89-97, ACM, May 1991.

28. Ruben Prieto-Diaz and Peter Freeman, ''Classifying Software for Reusability,'' *IEEE Software*, pp. 6-16, January 1987.

29. G. Salton, E.A. Fox, and H. Wu, ''Extended Boolean Information Retrieval,'' *Communications of the ACM*, vol. 26, pp. 1022-1036, 1983.

30. M.A. Shepherd and C. Watters, ''Virtual Structures for Hypertext,'' *Int'l Conf. on Multimedia Information Systems*, pp. 201-219, McGraw-Hill, 1991.

31. F.WM. Tompa, ''A Data Model for Flexible Hypertext Database Systems,'' *ACM Tr. on Information Systems*, vol. 7(1), pp. 85-100, January 1989.

32. Randall H. Trigg and Lucy A. Suchman, ''Collaborative Writing in NoteCards,'' *In: Ray McAleese (Ed.), Hypertext: theory into practice*, pp. 45-61, Intellect Inc., Oxford, 1989.

33. Y. Vassiliou, M. Jarke, E. Petra, T. Topaloglou, G. Spanoudakis, and C. Vezerides, *Technical Description of the SIB*, Institute of Computer Science, Foundation of Research and Technology - Hellas, Heraklio, Crete, January 1990.

34. C.V. Vezerides, ''The organization of an SIB for software reuse by a programming community,'' *Master's Thesis*, University of Crete, 1991.

35. N. Yankelovich, B.J. Haan, N.K. Meyrowitz, and S.M. Drucker, ''Intermedia: The Concept and Construction of a Seamless Information Environment,'' *IEEE*

Computer, pp. 81-96, January 1988.

36. P.T. Zellweger, ‘‘Scripted Documents: A Hypermedia Path Mechanism,’’ *Hypertext ’89 Proceedings*, 1989.

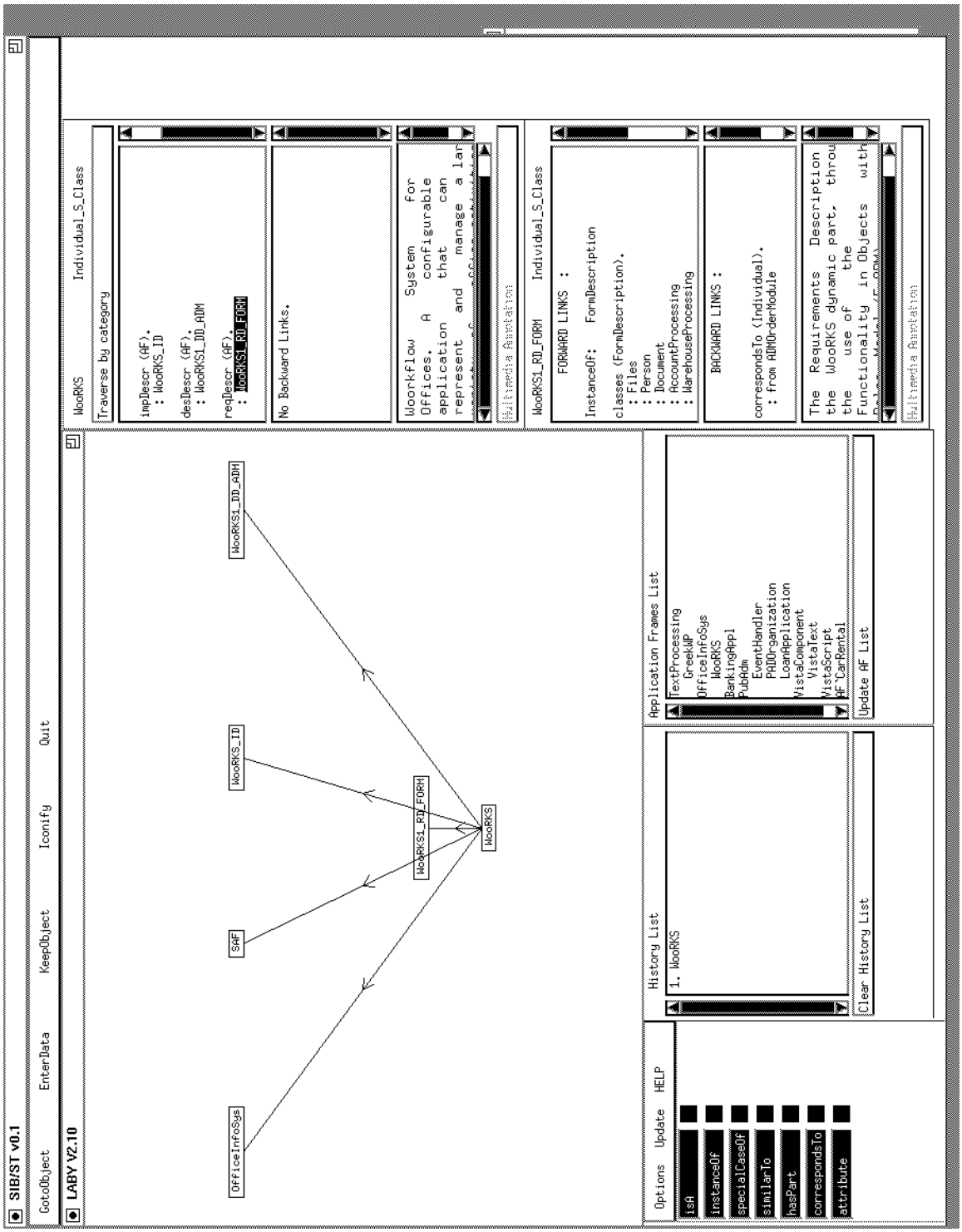

Figure 1. : The Selection Tool of the Software Information Base. Current in the Graphical Browser is WooRKS, the starting point of the usage example.

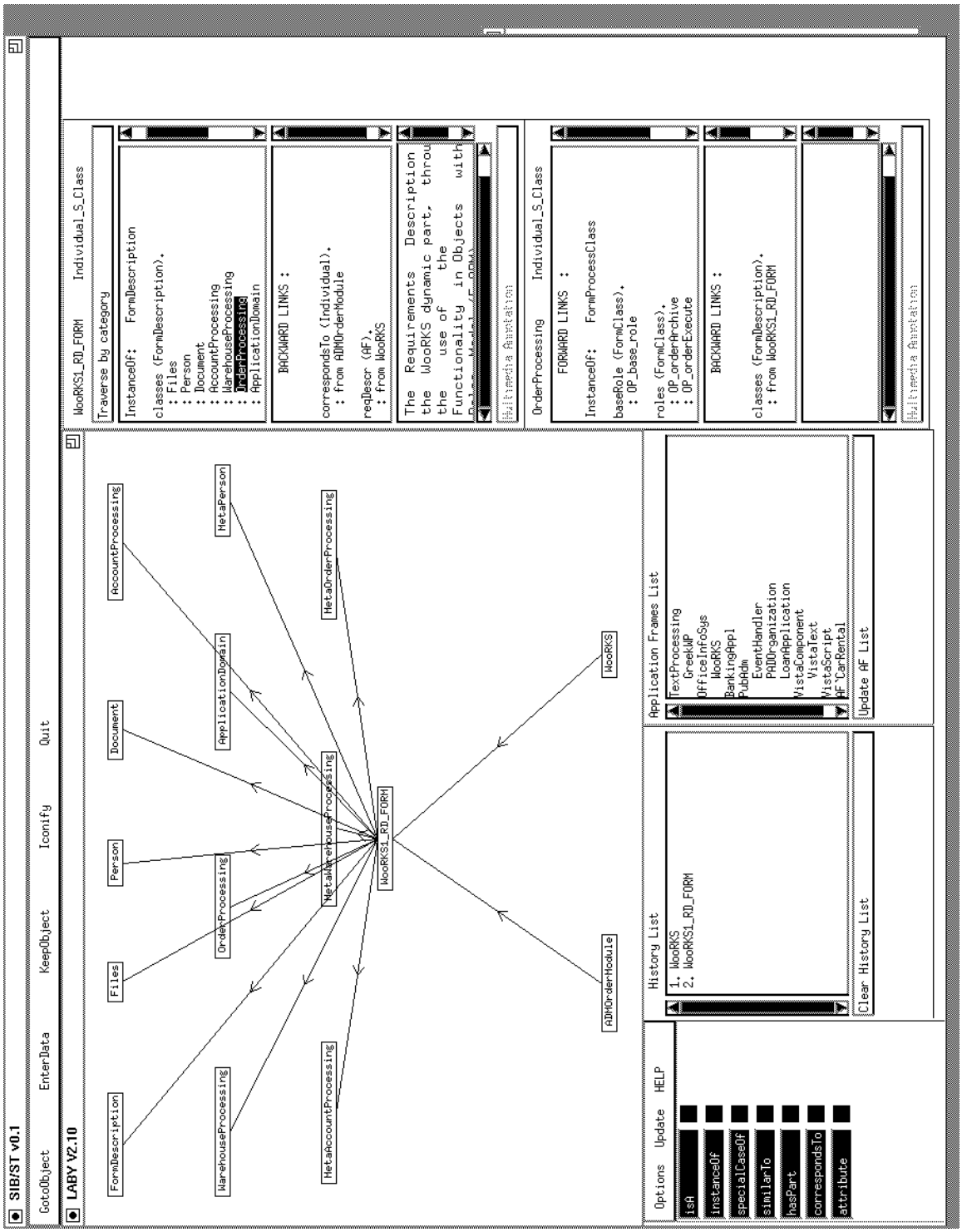

Figure 2. : Inspecting the requirements of WooRKS and previewing OrderProcessing.

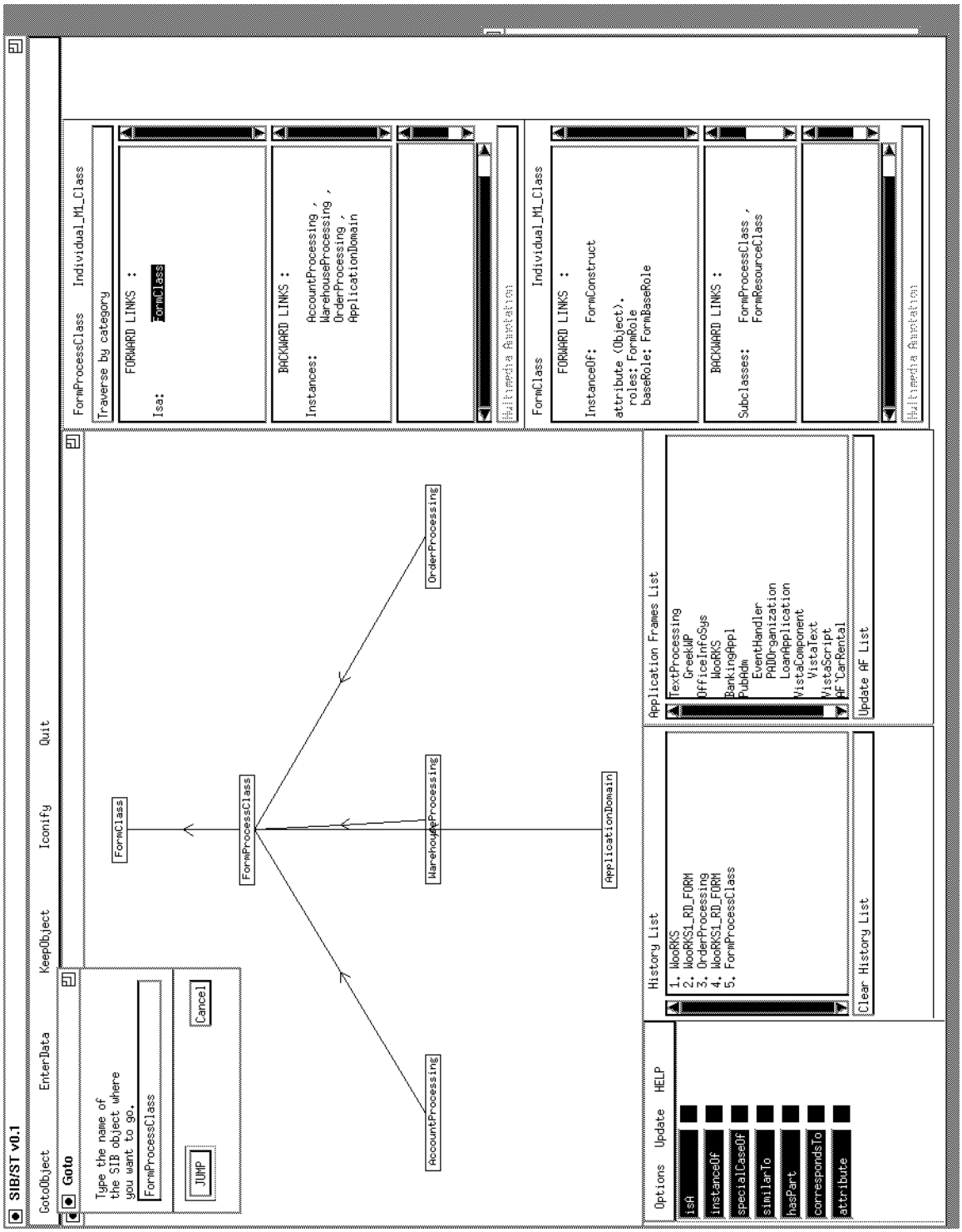

Figure 3. : Visiting FormProcessClass and previewing its superclass, FormClass.

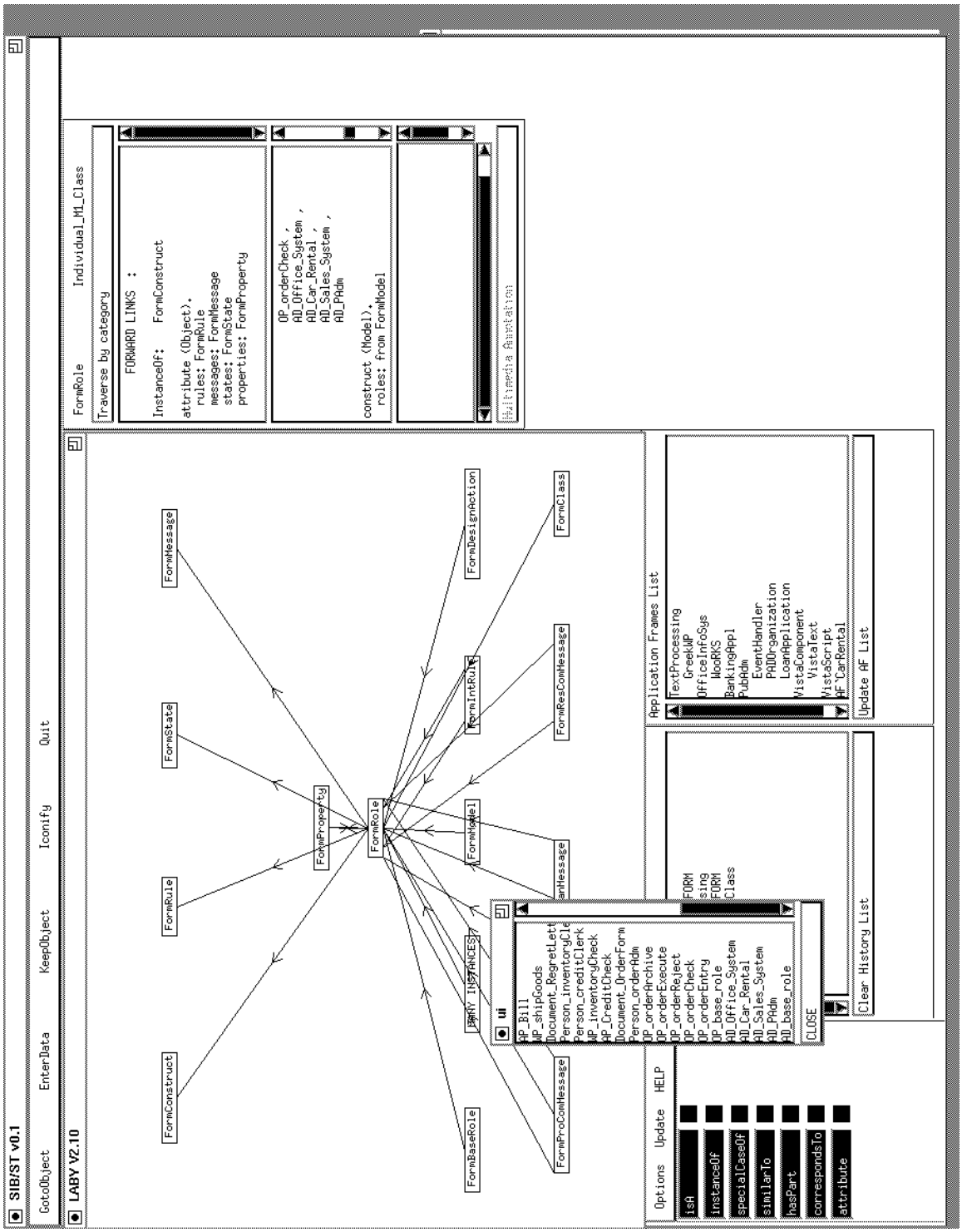

Figure 4. : The Instance Box handles the instances of FormRole.

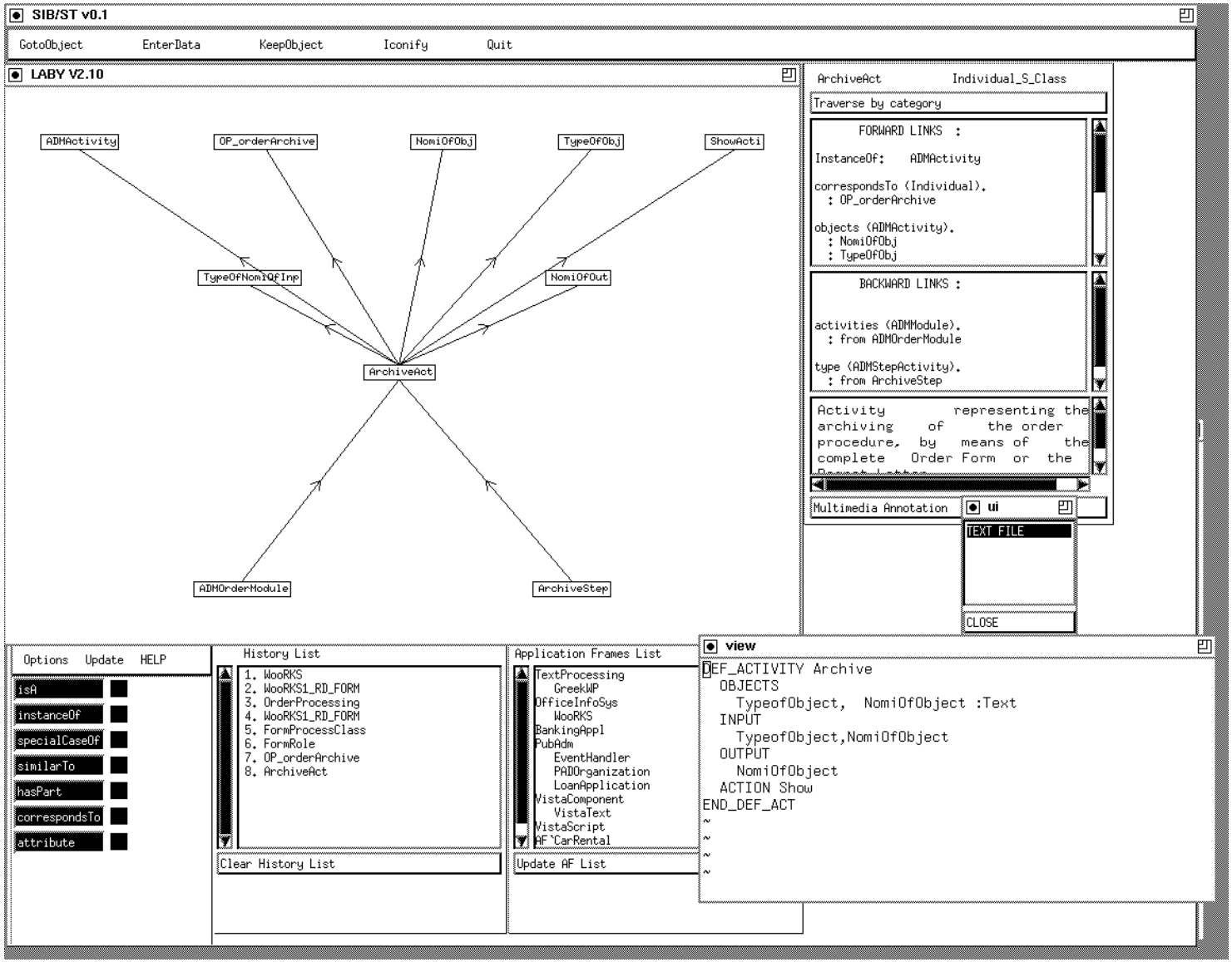

Figure 5. : An annotation attached to ArchiveAct.

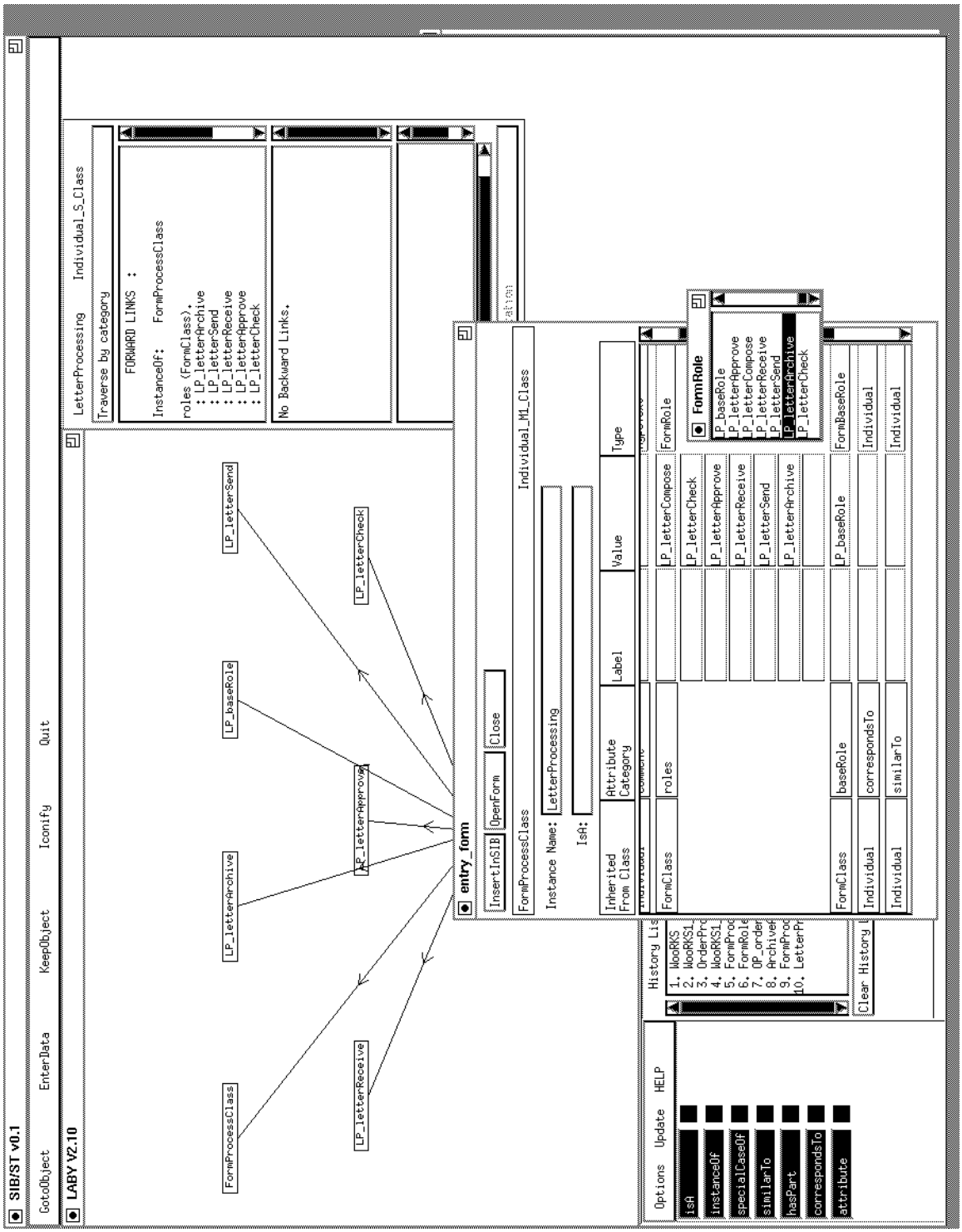

Figure 6. : The EnterData facility used to insert LetterProcessing in the SIB.

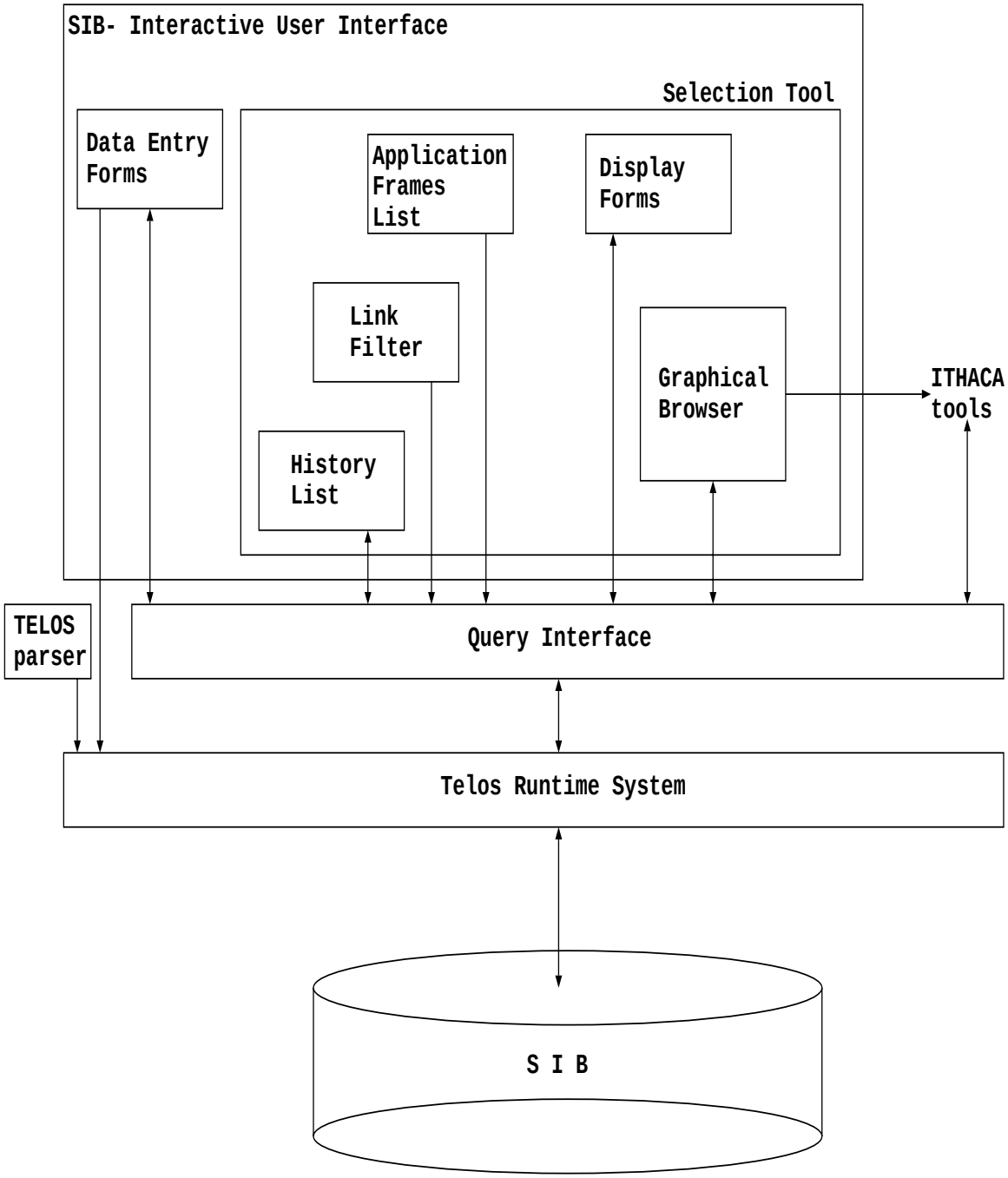

Figure 7. : The architecture of the SIB-ST