RECAST: A TOOL FOR REUSING REQUIREMENTS

M.G. Fugini §†, *B. Pernici* †

§Università di Brescia
†Politecnico di Milano,
piazza Leonardo da Vinci 32, Milano, Italy

Abstract

Reuse of development documents regarding application requirements makes the application development process more efficient and reliable. The REquirements Collection And Specification Tool (RECAST) being developed in the framework of the ESPRIT ITHACA project aimed at reusability under an object-oriented approach for Information System applications is presented in the paper.
Two types of application developers interact with RECAST: the Application Engineer, who maintains the knowledge about reusable components, and the Application Developer, who develops specific applications; their interaction with RECAST is presented. RECAST guides these developers using design knowledge stored in a Software Information Base (SIB).

1 Introduction

This paper describes RECAST (REquirement Collection And Specification Tool), a tool for reusing application requirements in development of Information Systems applications. RECAST is being developed as part of the ITHACA Project /Pro 89/, an ESPRIT II software environment research program encompassing European industrial and university organizations. The objective of ITHACA is to develop an advanced software development environment based on the object-orientation paradigm /Ara 88, Fis 87, Tsi 88, Tsi 89/ and on reusability. In particular, the *ITHACA Application Development Environment* (ADE) facilitates the reuse by the developers of various components of previously developed applications: application requirements, application designs, implemented objects, execution results, documentations /Gib 89b/. Consequently, the set of tools in the ITHACA Application Development Environment comprises requirement collection and specification tools, application configuration tools, object design tools.

The ITHACA Application development Environment is centered around the *Software Information Base* (SIB), a knowledge base storing information about available reusable components /Cos 89/. The SIB is accessed by the ITHACA tools in order to inspect available application components /Gib 89b/ in the context of the development of a new application to evaluate which existing components can possibly be reused totally or through refinement and modification.

Two types of designers are addressed by ITHACA tools: the *Application Engineer* (AE), and the *Application Developer* (AD) /ITH 89, Gib 89a/. The Application Engineer is

responsible for development of application skeletons, that is, sets of generic application components (application requirements, specifications, design objects, executables, design documents, etc.) meaningful to certain application domains and therefore candidates for reusability in applications pertaining to those domains. For example, in the "accounting system" application domain, reusable components are specifications and designs of "journal", "client", "account" objects, and operations on them like "update", "post", "withdrawal". These reusable components can then be expanded by the AD in different ways in order to develop specific applications. The Application Developer develops specific applications. Using the ITHACA tools, the AD's development paradigm consists of the definition of the application requirements, selecting reusable components from generic applications, and progressively *refining* and *modifying* these components to meet the requirements of the specific application being developed. The AD uses the tools of the ITHACA environment, maintained by the AE, to search for reusable components in the SIB, for configuring the current application reusing as many components as possible, and for implementing new objects. The product of a specific application development can be further evaluated by the AE for identifying new components that are candidates to reusability; these may become part of the available reusable software of ITHACA and therefore be described in the SIB.

In the ITHACA Application Development Environment, *reusability* of software and of development information is the major goal. This goal regards all the development phases, thus, in particular, the application *requirement collection and specification* phase. Such phase is conducted under a reusability approach that allows the AD to reuse existing *requirement specifications* and to identify reusable *design objects* that meet the requirements.

The purpose of this paper is to describe how the RECAST tool addresses these issues by supporting the AD in requirement collection and specification under an object-oriented paradigm, by automatically *completing* the requirements when existing specifications are discovered by the tool for reuse, and by giving *suggestions* about existing design objects that can be possibly reused.

The next section of the paper describes the rationale of the approach to reusability of application development documents in the first phases of the software development life-cycle. Subsequent sections present RECAST architecture (Section 3), the requirement collection phase (Section 4), the requirement specification phase (Section 5), and a scenario of use of RECAST (Section 6).

2 Reusability of development documents

The focus of research on reusability has mainly been on reusing code /Bur 87, Fre 87/. However, reusing the results of previous software projects is important in all development phases in the software development life-cycle. The importance of reusing results from the early development phases has particularly been emphasized: Freeman /Fre 87/ suggests the reuse not only of code, but also of specifications, and Feather

/Fea 87/ suggests to reuse specifications, modifying specifications rather than their implementations when change is needed. Reuse of algebraic specifications, built and structured in a modular manner to facilitate reuse, is presented in /Wir 89/.

RECAST may be regarded as a *system generator*, based on knowledge of given application domains and on models for requirement definition, with reusability of development documents as a primary goal. RECAST is based on domain knowledge specialized for development of Information Systems in different application domains. The support of information bases, specification bases, and knowledge bases for different phases of the development of software systems has been proposed in the literature /Tsi 89, Jar 89, Per 89b, Pun 87, Pun 88/.

The relevance for reusability of intermediate stable forms of development documents has been emphasized /Weg 87/. We assume an Information Systems development life-cycle based on the following phases /ITH 89/:

- requirement collection
- functional specification
- configuration of designs (implementations)

Each of these phases produces **development documents** based on the domain knowledge and on a model for that particular development phase. The requirement collection phase produces a Requirement Collection Document, the specification phase a Requirement Specification Document, and the third phase an implementation of the system in the target O-O language.

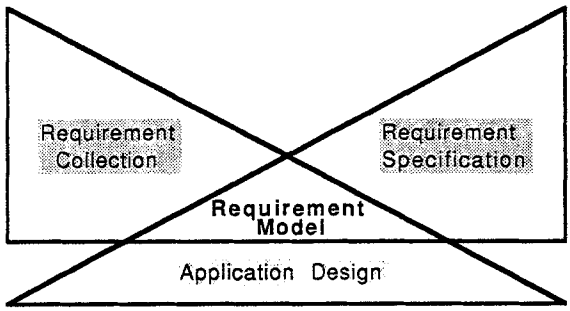
RECAST produces and reuses documents in the first two phases above. The structure and the contents of these documents is discussed in the following sections.

The definition of requirements in RECAST is oriented to building a system rather than to describing the relationship of the system with the external world. Therefore, the requirements define the system model, rather than the world model, the goal being that of describing the aspects that characterize a specific application in a given application domain. The specification documents define the functionalities of the components of the system. These components are defined by RECAST on the basis of available knowledge on a given domain and on the requirements described by the AD for the specific application.

We assume that the SIB contains both reusable building blocks and information about the development process. RECAST reuses elements in the SIB in three ways:

- *requirements are collected following a predefined domain model.* The domain dependent model allows the designer to define the characteristics of the application, using a domain dependent vocabulary, mainly on the basis of examples. Requirements may define not only functional characteristics of a specific application, but also non-functional characteristics such as distribution of users and data on different sites, volumes of data and processing frequency. The use of domain languages has been proposed in the literature for this purpose (e.g., in the Draco system

Figure 1: Relationship among Requirements Collection, Requirements Specification and Application Design through the Requirements Model



/Nei 87/); in RECAST, the domain model is inserted as knowledge in the SIB, for more flexibility (see Sect. 3).

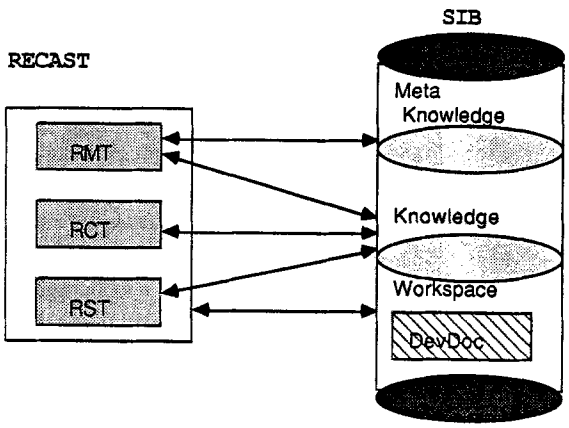
- *functional specifications* are written *combining specification elements*. These elements can specify default functionalities for a given domain, or be selected and composed (automatically or semi-automatically) from the SIB, according to the requirements defined by the AD. In addition, the AD may refine existing modules and create new elements, which may be made available for reuse in future projects, both as reusable elements and together with their development information. As suggested in /Fea 87/, refinement in RECAST is mainly based on adding to, and combining features of, existing elements, rather than changing predefined features.
- *design objects* (corresponding to implementations) may be *associated to specification elements*, and provide the basis for implementation of the O-O target application.

These three reuse categories of RECAST are strictly related to each other by a Requirement Model (Fig. 1). The *Requirement Model* (RM) contains relationships between requirement, specification and design components, information about their structure, and about tools that can be used to define these components. Additional information describing the components and their relationships is also contained in the SIB with the purpose of facilitating their reuse.

In general, reusable components, although constructed by developers under a reusability approach, are maintained and inserted in the RM by the AE in order to control and guarantee the quality of the model.

The AD is guided through RECAST in the definition of the requirements of the specific

Figure 2: Architecture of RECAST



application using information contained in the RM. A set of tools is provided within RECAST (or called by RECAST) to help the AD in producing the requirement and specification documents for a specific application, as described in the following sections.

In RECAST, the domain model and language used for requirement collection may vary for different applications and different domains, while the specification language is an internal language of the tool, domain independent, and defined according to an extended object-oriented paradigm /Per 90/. The O-O paradigm in specifications facilitates the definition of standardized parts in the specifications and of interfaces between them. In fact, the description of object interfaces has been used for system specifications by many authors /Mey 89, Cox 87/. Moreover, in the Ithaca development environment, the goal is that of developing O-O applications. Therefore, while we keep the requirement collection phase open to any other approach, we suggest that the specification phase is chosen as the borderline between models used by developers during the requirements collection phase, based on domain dependent concepts, and the O-O world.

3 Architecture of RECAST

The architecture of RECAST is shown in Fig. 2; the tool is composed of the *Requirement Modeling Tool* (**RMT**), the *Requirement Collection Tool* (**RCT**), and the *Requirement Specification Tool* (**RST**). The three tools are centered around the knowledge provided by the *Requirements Model* (RM), which groups knowledge about the application domains and about the phases of requirement definition, together with a

link to possible implementations, as shown in Fig. 1 and 2.

RECAST, with the RMT, assists the AE in filling in and modifying the RM, stored in the SIB. Through the RCT and RST modules, RECAST assists the AD in producing a *Development Document* (**DevDoc**) for a specific application, composed of a *Requirement Collection Document* (**RCDoc**) and of a *Requirement Specification Document* (**RSpecDoc**). The DevDoc is the basis for the subsequent development phases of ITHACA, that is, the phases regarding configuration of designs and implementations. These phases are carried on by the AD using the other tools of the ITHACA Application Development Environment, such as the Visual Scripting Tool, which supports application configuration out of existing objects, and the Object Design Tool, which supports the design of new objects in the target O-O development language /ITH 89/. Requirement collection and specification are performed in RECAST through a set of steps and using a notion of "unit" for consistency reasons. In particular, collection is performed through a process of production of *Collection Units* (CUs); these units are then organized in a structured RCDoc, which is part of the output of RECAST. Analogously, requirement specification is a process of production of *Specification Units* (SUs) which are then organized by RECAST in the structured RSpecDoc.

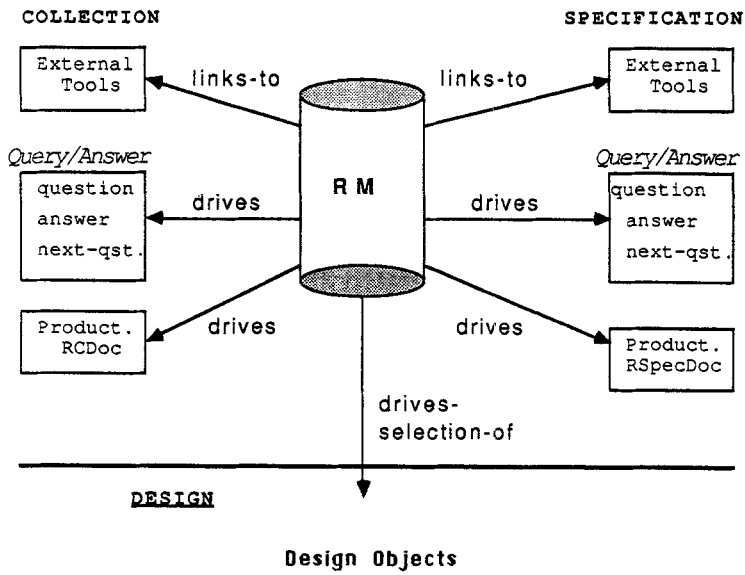
The RM is stored as a semantic network in the ITHACA SIB knowledge base. This network contains:

- knowledge about the requirement models, about application domains, and about the process of defining in RECAST new application domains and reusable components with the related development assistance rules ("meta-knowledge").
- knowledge about the requirement collection and specification process used in ITHACA ("knowledge").

The interaction of RECAST with the developers is obtained through *navigation* in the knowledge base using the RM (see Fig. 3) for:

- guiding the interaction with the developers ("dialogues") and interpreting the user choices ("answers");
- allowing the developer to use available *external tools* necessary to perform some of the activities related to requirement production. These can be general-purpose tools, such as editors, text formatters, graphical packages or interface managers, or special-purpose tools, for example application document generators and managers. These tools are linked to the RECAST environment and can be called by the user to perform some functions. Such links provide flexibility to RECAST because tools can be easily integrated in the requirement definition environment by the AE by simply adding to the knowledge of RECAST a link to the tool. In current work on RECAST, specialized tools that scan a document and derive its conceptual structure /Per 89a/ are being linked to RECAST;
- producing the RCDoc and the RSpecDoc;
- inspecting the set of available design objects to identify reusable objects, i.e., objects that meet the current requirements; acknowledgment about the existence

Figure 3: Role of the RM in RECAST



of such objects is inserted by RECAST into the DevDoc in the form of design suggestions.

In this paper, we focus on the description of the contents of the SIB for use by RECAST (Sect. 3.1. describes the designer working area, Sect. 3.2. and 3.3. the meta-knowledge and knowledge) and on the RCT and RST modules.

3.1 The developer workspace

The AD is provided by RECAST with a workspace where:

- information is moved from the SIB to be examined;
- external tools operate when invoked through RECAST;
- requirements are inserted as part of the current application requirements in two ways:
  a) upon creation by the AD
  b) automatically by RECAST, when reusable requirements are found in the SIB.

The requirements are transferred, created, deleted from the workspace according to CUs and SUs which are the atomic entities of consistency of RECAST and of the workspace.

A sample image of the workspace for the AD is shown in Fig. 4.: the structured RCDoc

Figure 4: Image of the AD workspace

```
-----------------------------------------------------------------
STRUCTURED REQUIREMENTS COLLECTION DOCUMENT

     RCDoc = set-of CU
     CU = set-of REQ
     REQ = Question + Answer Set + RECAST annotations

STRUCTURED REQUIREMENTS SPECIFICATION DOCUMENT

      RSpecDoc = set-of SU
      SU = set-of SPEC
      SPEC = Object Specification +
             Design Suggestions +
             links to CUs
-----------------------------------------------------------------
```

is progressively built as a set of CUs; analogously, the RSpecDoc is built as a set of SUs. A CU is a set of requirements (REQ in Fig. 5) derived by posing questions to the AD and by interpreting the user answer within an answer set. Additionally, REQ contains annotations given by RECAST about requirement collection, that is, about missing requirements that are automatically added by the tool reusing available CUs. In the specification phase, SUs are built as sets of specifications. A specification (SPEC in Fig. 4.) is derived by RECAST as a set of specification objects and as a set of design suggestions. Design suggestions are links to reusable design objects. In each SPEC, RECAST introduces a link to the requirements that lead to the generation of the SPEC.

3.2 Semantic network for requirements modeling

Requirements modeling in RECAST is based on a semantic network containing both meta-knowledge (generic knowledge for model construction) and knowledge (knowledge for modeling applications in a given application domain).

The semantic network that is used mainly by the AE for management of the RECAST environment is shown in Appendix 1. The meta-knowledge represented in this semantic network regards the task of the AE of defining a given RM. The definition of a RM consists of the definition of the **model characteristics**, such as the application domains where the model can be applied, whether the model data-oriented (e.g., an ER model) or process-oriented (e.g., Petri nets), and whether external tools exist for model processing (e.g., a graphical editor for entering the model entities).

The next component of the RM definition process is the definition of the **model entities**

(**ENTITY** node in the semantic network). An entity, identified by its name, has a representation (e.g., graphical), and some attributes.

The definition of an RM then encompasses the definition of the **model links**, i.e., of the relationships among entities (**LINK** node). A link, identified by its name, has some **CHARACTERISTICS**: origin and destination entities (this also defines the link direction), and a type. The type can be standard, such as "is-a" or "part-of" link, or can be defined by the AE. In the network, the "is-a" links are used as a classification mechanism and the "part-of" links are used for defining the components of a node. No inheritance mechanism holds in the network. External tools mentioned earlier in this section are connected to LINK nodes; this mechanism allows the AD or AE to be aware of the availability of external tools while traversing links.

Navigation of the AE through this meta-knowledge occurs according to the modalities that will be illustrated in the next section. The purpose of navigation for the AE is to be guided in the maintenance of the environment; besides the introduction of new models, the AE navigates for exploring the base of reusable requirement elements, for creating connections among requirements, specifications, and designs, for defining application domain knowledge and organizing it in structured hierarchical subdomains. The AE is also responsible for linking external tools to the environment and for defining the questions and answers associated to links.

In Appendix 2, a sample semantic network of RECAST knowledge is depicted. This network describes the knowledge used by RECAST to guide the requirements collection and specification by developers in specific applications and it is constructed according to RECAST meta-knowledge, in terms of ENTITY nodes, LINK nodes, and their characteristics. The network shows, as an example, the Requirement Definition Process in the "Public Administration" application domain. It is assumed that an AD is collecting the requirements of an application in this domain and that a definition process based on a model (PADM model in App. 2 a) exists especially oriented to developing applications in the Public Administration Domain. A method for application development in the Public Administration domain is being used in ITHACA as a common ground for evaluating the functionalities of the various ITHACA tools /Kap 89a, Kap 89b/. The application being developed is a system for a City Council; this system automates the procedure of releasing authorizations to private organizations to hold public events (concerts, exhibitions, etc.). App. 2 b) shows the information needed for requirements collection and App. 2 c) the information needed for requirements specification. The use of the knowledge illustrated in App. 2 is described in detail in Sect. 4 and Sect. 5., and an example of RECAST use based on this knowledge is shown in Sect. 6.

3.3 RECAST interaction with the developers: links and navigation on the semantic networks

RECAST guides the Application Developer and the Application Engineer in navigating along nodes and links of the semantic networks that constitute its base of knowledge,

performing actions and calling available external tools associated to network links. *Link traversal* is the basic mechanism for generating queries to the developers and for calling external tools during specific tasks of the requirement definition process.

Default queries can be automatically associated to standard links ("is-a" and "part-of"). The default query:

```
YOU CAN SELECT ONE OF THE FOLLOWING DEFINITIONS:
   list of choices
```

is associated to "is-a" links; the developer is requested to select among possible choices listed by the tool. The choices are derived by RECAST from the semantic network, which describes "is-a" relationships between elements of the model.

The following default query:

```
WHICH PART DO YOU WANT TO SELECT?
   list of choices
```

is associated to "part-of links"; the developer is requested to select among possible components linked with a "part-of" link in the semantic network.

Optionally, links may have associated **questions** for interaction with the developer during requirements collection and specification according to the models defined by the Application Engineer. To each link type, an **answer set** is associated, which contains information to elaborate the user answers. An **inverse question** may be associated to links for their traversal in inverse mode. Questions attached to links may refer to tools (EXTERNAL TOOL node in App. 1.) which can thus be called and used from within RECAST.

Links have a **state**. The state can assume the following values:

- *potential*: the link is not visible to the AD because it does not belong to the currently selected path, (e.g., because the AD is working within a given application domain and therefore a sub-network has been selected);
- *active*: the link is visible to the AD because it belongs to a currently selected path; potential and active links provide the Application Developer with a view mechanism on the semantic network;
- *selected*: links can be selected by the AD when they are relevant to the current application and moved to the AD's workspace; with this mechanism a view on the semantic network is defined for a specific application (designer workspace for the specific application).

All the links in the semantic networks are potential, that is, can be traversed. Some of them become active, that is, can be traversed, depending on answers to previous

queries; for example, if the AD decides to use the Entity-Relationship model for defining data schemas in the "accounting systems" application domain, the links necessary to create CUs and SUs according to the E-R model and according to available reusable requirements in that domain become active.

The semantic network can be traversed in two modes:

- retrieval mode
- update mode

In the first mode, the AE or the AD explore the RECAST knowledge or the AD's workspace in order to find concepts useful in the development or to examine the current contents of the workspace. The update mode occurs when concepts are selected from the network and information is copied from the SIB into the workspace. In retrieval mode, traversal of active links prompts the corresponding queries; no actions are undertaken on the SIB, but each link traversal operation activates other links. In update mode, the selection of a link determines insertion operations on the workspace.

4 Requirements collection tool

The Requirements Collection Tool is realized in RECAST using the navigation mechanism and knowledge in the semantic network concerning requirements collection. Requirements collection is carried out by progressively creating Collection Units (see App. 2a and 2b) composed of *functional* and *non functional* requirements. Functional requirements regard the application entities and procedures, that is the functionalities of the application; non functional requirements regard the application interface, the hardware and software characteristics of the application, the organization, and a variety of quantitative parameters and constraints (such as, expected response times, data volumes, computer loads). Some examples of non functional requirements are shown in App. 2b.

The output of requirements collection is the structured RCDoc. This document is composed of a set of CUs which are organized by RECAST in the RCDoc through some *Structuring Rules*. Such rules determine where collected information has to be placed in the RCDoc and how this document can be accessed in order to find "related" information. CUs are produced by RECAST according to the Q/A paradigm for requirements collection illustrated in the previous section.

The remainder of this section describes navigation of RECAST through the semantic network of App. 2. for: setting the AD requirements definition environment for the City Council application (Sect. 4.1.); navigating in the network along specific knowledge referring to the preparation of CUs according to the selected method (Sect. 4.2.).

4.1 Setting RECAST environment for the AD

RECAST drives the AD in setting a collection and specification environment, that is, in selecting the adequate knowledge for the application at hand. Setting the environment is achieved with RECAST navigating along the knowledge and selecting a model and the application domain for a given specific application to be developed. In the network of App. 2 a), three types of models are shown: Structured Analysis, PADM model /Kap 89a/, and SADT & ER models /Som 89/. Application domains have an associated size and a description that brings to three subdomains: the Public Administration domain, the domain of Chemical Applications, and the domain of Financial Applications. In our case, the AD selects the Public Administration domain, which is composed of a set of phases: each phase produces a set of CUs; CUs are in turn possibly made of CUs.

4.2 An example of requirements collection knowledge

In the network of App. 2 b), the knowledge associated to the production of CUs within the Public Administration domain with the PADM method /Kap 89a/ is illustrated (only active links are shown). The network shows the representation of the requirements model developed for the PADM method. A CU is made of entities (functional requirements) and of non functional requirements. The basic entity of this model describes procedures and is called *case type* /Kap 89a/; a case type is composed of a schema (flowchart) of *steps*. Steps of a case type have associated *documents*; we suppose that the reusable base of documents contains three document types: the OFFICIAL DOCUMENT, the REQUEST document, and the APPROVAL document. Each type of document has a default form which is shown to the AD: the AD decides whether such form is suitable to his purposes and can be reused. This is shown in App. 2b for the REQUEST only. The REQUEST document has a default definition; alternatively, an example of REQUEST can be entered by the AD and one of RECAST external tools can be called (e.g., the INTRES tool which understands document structures by examples /Per 89a/) or entering an example. Depending on which link is traversed in the network, the proper external tool is called by RECAST into the AD workspace. A case type also comprises the definition of *agents* who represent business roles involved in the use of the application. Here, an agent can be an EXTERNAL OFFICE (of the organization who requires the authorization to the public event) or an OFFICE of the City Council.

A tool which facilitates the reuse of existing elements in the description of case types has been defined /San 90/; the tool allows the definition of case types by example, based on available case components; defined examples are generalized and aggregated by the tool in order to define case types.

5 Requirements specification tool

The goal of requirements collection and specification in ITHACA is to enable the developer to reuse as many existing specification components as possible. We have seen how RECAST guides the developer in requirements collection, helping him to identify and to choose from, or to define, predefined reusable components. In a similar way, functional specifications are constructed from predefined specification elements stored in the SIB.

In the following section, we discuss how specification elements are retrieved from the SIB, composed in specification units and linked to design objects (implementations) by the RST module of RECAST.

5.1 Requirements specification units

The Requirements Specification Document (RSpecDoc) describes the functional specifications of a specific application. The requirements specification document is composed of a set of specification units (SUs). Specification units describe specification components or composition of specification components. The requirements specification tool (RST) uses the navigation mechanism provided by RECAST, as in the case of the RCT. Contrary to the case of the requirements collection phase in RECAST, the model used for requirements specification is an internal model known to RECAST, common to all application domains. This choice is justified by the necessity of developing a homogeneous specification components base, providing components which are reusable across application domains. The internal specification model is based on an extended O-O paradigm: the Objects with Roles Model (ORM) /Per 90/. An object-oriented paradigm has been chosen for requirements specification in ITHACA for two main reasons: the object-oriented paradigm provides abstraction and encapsulation constructs that make definition of specifications and their composition easier; the target development environment in ITHACA is object-oriented, therefore specifications at all levels and designs are performed according to this paradigm. The ORM model provides a high level representation model for objects: object properties and methods are partitioned using the concept of role. An object interface may be different according to the different roles that the object may perform, and the internal state of the object. Within a role precedences of application for methods and constraints may be defined with transition and constraint rules (the reader interested in a detailed description of ORM is referred to /Per 90/).

Specification components are stored in the SIB in form of ORM pre-defined objects and their components (roles, messages, properties, states and rules). Information associated to the semantic network instructs the tool to support the mapping from requirements collected according to domain specific models to specifications units defined according to the RECAST internal specification model.

In the following section, we discuss how specification elements are retrieved from the

SIB and composed in specification units.

5.2 Derivation of functional specifications

The Application Developer is assisted by RECAST in deriving requirements in three ways:

- providing *default objects.*
  For deriving object descriptions, we may assume that a number of requirements is implicit in a given application domain; in these cases, there is no need to ask the Application Developer to collect these requirements, and the tool for requirements specification is able to complete the collected requirements with application domain dependent default assumptions. For instance, in the Public Administration domain it is obvious that some document preparation functions must be provided. Therefore, there is no need to ask the AD if these functionalities are needed, rather it is necessary instead to collect requirements about the quality of the documents to be produced, the volume, the characteristics of the secretarial personnel, the security and access constraints, and so on. Basic functionalities, such as editing, formatting, printing documents are not to be explicitly stated and are inserted in the specification document by default.
- *(semi-) automatically deriving ORM objects.*
  We assume that collection units and specification units can be either in interpreted or non-interpreted form /Gib 89a/. Interpreted units refer to sentences in a language whose syntax is known by RECAST, non-interpreted units store development results, independently of the model used to develop them (e.g., free text, or non-interpreted diagrams and charts). The mapping rules are used to support the developer in mapping from collected requirements to specifications; an automatic (or semi-automatic) mapping can only be performed for units in interpreted form. A mapping has the following results:

  - new specification units are created
  - active links are created from collection units to specification units; each of these links has attributes defining its query/answer interface, tool invocation, as described for the next-phase links.

  A mapping may be performed using a number of techniques, with the goal of combining collection units and selecting from the specification base in the SIB the appropriate specification units. Fig. 5.a shows the structure of mapping rules, which define, corresponding to a CU, which are the possible corresponding SU. Mapping rules are useful when the previous collection phase has been performed mainly in a guided way, thus yielding interpreted requirements. Mapping rules are associated to links in the semantic schema, which are activated when the "next-phase" link is traversed from a PHASE entity (see App. 2a) to the next one.

Figure 5: Mapping rules

a) Mapping rule from requirements to specifications

```
REQUIREMENT UNIT -----> ORM OBJECT, ROLE, OPERATION
```

b) Mapping rule from requirement specifications to designs

```
{ORM OBJECT set}, {ROLE set}, {OPERATION set} ----->
DESIGN OBJECT, DESIGN ANNOTATIONS
```

- *refinement by the developer.*
  The developer must complete the definition of specification units, until all collection units are mapped into specification units. In particular, non-interpreted requirements have all to be mapped into specifications manually. In some cases, also interpreted requirements may require manual refinement.

The mapping from requirements to specifications according to the ORM model is represented in the RM following the schema of App. 2 c). Entities and links of the semantic network of the RM are handled as in the requirements collection part of RECAST, that is:

- some links and entities are predefined for a given application domain by the Application Engineer. In particular, some transformation tools are associated to some of the links; the mapping tool is associated to the "next-phase" link.
- specialization of basic components may be created by the Application Developer in two ways:

  1. some entities and links are automatically created or activated by the mapping tools.
  2. some entities and links may be created by the Application Designer as a refinement of pre-existing entities.

ORM design tools can be called traversing appropriate links in the semantic network.

5.3 Design suggestions

Design suggestions for pre-defined specification units are automatically provided through the Requirements Model. As with the mapping rules presented in the previous section, these associations may be performed dynamically, through domain-dependent rules contained in the RM, taking into consideration several aspects of the requirements, the domain knowledge, and existing specifications (Fig. 5.b).

Design suggestions are basically of two types:

- object class names
- annotations about suggested implementation strategies (for instance, references to previous implementations, similar implementations, possible basic components for the implementation).

Some examples of mapping from requirements to specifications, including some design suggestions, are presented in the following section, describing a scenario of use of RECAST.

6 Scenario of use of RECAST

In this section, we describe a scenario of use of RECAST, calling also some external tools for requirements specification.

The scenario is described illustrating an example dialog with an AD interacting with RECAST. The example is taken from the Public Administration application domain /Kap 89a/.

Collection phase

First, the developer sets up the appropriate environment answering questions about the general characteristics of the application at hand. We assume that the developer will have to answer these questions only the first time a session is started. We do not show here the dialog for setting up the environment, but we focus on the collection of requirements about functionalities in a specific application. As presented in Sect. 3., queries are formulated according to the semantic network of App. 2. We assume that the developer is using the method for the Public Administration domain (PADM in App. 2a). This example of interaction shows how queries are composed automatically by RECAST, directly from the structure of the semantic network, and from information associated to links. Referring to App. 2b, an example of dialogue session is reported in Fig. 6., where a star (*) marks selected options.

The dialogue shown in Fig. 6. shows how a collection unit is composed. An entity is defined, in particular a case type. A case type is being defined. The name "approval" is assigned to the case type. A case type has a schema of steps, and the step "preparation" is being defined. Since the STEP is a reusable component, RECAST allows the AD to see available steps (TYPICAL DEFINITION in App. 2b): additionally, a tool is available to define steps by examples (associated to the "is-a" link from the EXAMPLE OF STEP entity to the ENTER EXAMPLE entity in App. 2b) /San 90/.

We suppose that the AD has selected the DOCUMENT entity associated to the STEP, selecting the "has-part" link from STEP to DOCUMENT. The option of defining a specialization of an entity in the semantic schema is always present. In the example,

```
PUBLIC ADMINISTRATION HAS COLLECTION UNITS

COLLECTION UNIT
  WHICH PART DO YOU WANT TO SELECT?
  *   ENTITY
      NON-FUNCTIONAL

ENTITY
  YOU CAN SELECT ONE OF THE FOLLOWING ENTITIES:
  *   CASE TYPE

CASE TYPE
  CASE NAME:
      approval

  WHICH PART DO YOU WANT TO SELECT?
  *   SCHEMA OF STEPS
      DOCUMENT

SCHEMA OF STEPS
  WHICH PART DO YOU WANT TO SELECT?
  *   STEP
  YOU CAN SELECT ONE OF THE FOLLOWING ENTITIES:
      STEP DEFINITION

STEP
  STEP NAME:
      preparation

  WHICH PART DO YOU WANT TO SELECT?
      FOLLOWS
      AGENT
      STATE
      ACTION
      SELECTION CONDITION
  *   DOCUMENT

DOCUMENT
  YOU CAN SELECT ONE OF THE FOLLOWING DOCUMENTS:
  *   REQUEST
      APPROVAL
      OFFICIAL

REQUEST
  WHICH PART DO YOU WANT TO SELECT?
      DEFINITION
  *   DEFINE SPECIALIZATION

REQUEST SPECIALIZATION
  NAME:
       police-doc

ATTRIBUTES:
       none

POLICE-DOCUMENT
  *    DEFINITION

DEFINITION
  YOU CAN SELECT ONE OF THE FOLLOWING DEFINITIONS:
  *     EXAMPLE
        FREE TEXT

EXAMPLE
  YOU CAN SELECT ONE OF THE FOLLOWING EXAMPLES:
        TYPICAL DEFINITION
  *     ENTER EXAMPLE

PLEASE ENTER THE EXAMPLE

    ........
```

Figure 6: Example of dialogue driven by RECAST

the AD is defining a new document type "police-doc". After definition of the new document type, control of the dialogue returns to the pre-defined semantic network; the new defined document type is entered and considered as a regular entity in the semantic schema (see dashed box in App. 2b). An example of REQUEST document may be entered (a tool for entering document definitions, such as an editor or a scanner, is called by RECAST at this point).

Specification phase

With the same mechanism shown for the requirements collection, it is possible to change phase ('next-phase' link), and prepare the specification units corresponding to the collection units built with the illustrated dialogue. In the specification phase, mapping is performed directly by the AD and is assisted by mapping rules.

Figg. 7. and 8. provide a simplified version of the City Council application using RECAST for specification of requirements.

Fig. 7a illustrates the existing predefined ORM specification objects. An ORM object has a name, a set of roles, and, for each role, a set of applicable operations (shown in curly brackets). Only the principal characteristics of each object are shown. Fig. 7b depicts the default ORM objects provided by the AE in the Public Administration domain; some of these objects are usually taken globally, such as the "document" and "official-document" objects, while other objects may be selected considering only the roles relevant in the specific application.

Fig. 8. shows some mappings from requirements to specifications in the Public Administration domain. In Fig. 8a, some general mapping rules are shown. In Fig. 8b, the actual mapping from application requirements in the City Council example to specifications is shown. Note that some of the derived roles appearing in Fig. 8b are derived automatically from default objects and/or from mapping rules; other roles have been selected from the SIB and added to the specifications by the developer, or deleted from suggested specifications.

7 Concluding remarks and future work

Reusability of requirements is one of the goals of the ITHACA Esprit II Project; in this paper we have illustrated the basic features of the RECAST tool that is being implemented in the ITHACA framework for reusing development documents related to requirements.

The approach to reusability of requirements on which RECAST is based is a guided collection of the requirements of a specific application, together with a from these requirements to specifications reusing as many available specification elements as possible. In guiding the collection of requirements and in supporting the selection of suitable specifications, RECAST uses knowledge about requirement models and about application

Figure 7: Predefined components

```
            a) PREDEFINED ORM OBJECTS

person/office
    roles: request-handler
           reminder/informer
           document-preparer
           asker-for-approval
           approver

document
    roles: been-prepared
              role-messages: {input, visualize/retrieve}
           been-delivered
              role-messages: {print, archive, send}

official-document
    roles: been-prepared
              role-messages: {sign}

external-office
    roles: approver
           been-informed


   b. PUBLIC ADMINISTRATION DEFAULT SPECIFICATION OBJECTS

document

official-document

external-office
    roles: been-informed

person/office
    roles: document-preparer,reminder/informer, asker-for approval
```

Figure 8: Mappings

a. MAPPING RULES

Collected Requirement	Specification components to be selected: ORM object	role
office	person/office	
office-director	person/office	approver
request	official-document	all roles

b. MAPPING OF EXAMPLE REQUIREMENTS

Requirements	Specifications ORM object	role
office	person/office	document-preparer, reminder/informer, asker-for-approval, request-handler
office-director	person/office	approver, reminder/informer
	document	been-prepared, been-delivered
	official-document	been-prepared,been-delivered
	external-office	been-informed

domains stored in a knowledge base (Software Information Base).

Connections of RECAST with the other tools of the ITHACA Application Development Environment are being studied. Experiments on sample applications are currently being performed, based on a ground of implemented objects in a given application domain (the Public Administration application domain). Ideas on reuse of requirements are being validated on these objects using the ORM specification model; work on refinement of this model is also being done.

Acknowledgments

We acknowledge the contributions of the ITHACA partners in the Tools Group through discussions and comments.

This work was partially supported by a research contract between Datamont and Politecnico di Milano within the ESPRIT II project Ithaca (Project N. 2121) of the Commission of European Communities. We also acknowledge the contribution of the participants to Project "Informatica e Calcolo Parallelo - Obiettivo Infokit" of the Italian National Research Council through discussions on the topics presented in this paper.

References

/Ara 88/ G. Arango, R. Cazalens, and J.-C. Mamou, "Design of a software reusability system in a object-oriented environment", *Rapport Technique Altair 25-88*, Nov. 30, 1988.

/Bur 87/ B.A. Burton, R.A. Wienk, S.A. Bailey, et al., "The reusable software library", *IEEE Software*, July 1987.

/Cos 89/ P. Costantopoulos, M. Jarke, J. Mylopoulos, B. Pernici, E. Petra, M. Theodoridou, and Y. Vassiliou, "The ITHACA Software Information Base: Requirements, Functions, and Structuring Concepts", in /ITH 89/.

/Cox 87/ B. Cox, *Object-oriented programming*, Addison-Wesley, 1987.

/Fea 87/ M.S. Feater, "Reuse in the context of transformation based methodology", in /Fre 87/.

/Fis 87/ G. Fisher, "Cognitive view of reuse and redesign", *IEEE Software*, July 1987.

/Fre 87/ P. Freeman, *Tutorial: Software Reusability*, IEEE Computer Society, 1987.

/Gib 89a/ S. Gibbs, V. Prevelakis, D. Tsichritzis, "Software Information Systems: A Software Community Perspective", in /Tsi 89/.

/Gib 89b/ S. Gibbs, G. Kappel, "The ITHACA Application Development Environment - Process Models and Tools Scenario", in /ITH 89/.

/ITH 89/ ITHACA Tools Group, "Tools Group Interim Report", July 1989.

/Jar 89/ M. Jarke, DAIDA Team, "DAIDA: Conceptual Modeling and Knowledge-based Support" (draft version), Sept. 1989.

/Kap 89a/ G. Kappel, "Proposed reference example for the TWG in ITHACA", ITHACA.CUI.89.E. (Revised Version), Sept. 1989.

/Kap 89b/ G. Kappel, "Reusable software components for application of the Public Administration domain", ITHACA.CUI.89.E.#12, Sept. 1989

/Kou 89/ M. Koubarakis, J. Mylopoulos, M. Stanley, M. Jarke, "Telos: A knowledge representation language for requirements modelling", Int. Rep. KRR-TR-89-1, Univ. of Toronto, Jan. 1989.

/Mey 89/ B. Meyer, *Object-Oriented Software Construction*, Prentice-Hall Intl. Series in Comp. Sc., 1989.

/Nei 87/ J.M. Neighbors, "The Draco approach to constructing software from reusable components", in /Fre 87/.

/Per 89a/ B. Pernici, G. Vaccari, R. Villa, "INTRES: INTelligent REquirements Specification", Proc. IJCAI Workshop on Automating Software Engineering, Detroit, Aug. 1989.

/Per 89b/ B. Pernici, F. Barbic, M.G. Fugini, R. Maiocchi, J.R. Rames, C. Rolland, "C-TODOS: An automatic tool for office systems conceptual modelling", ACM Trans. on Information Systems, Oct. 1989.

/Per 90/ B. Pernici, "Objects with Roles", ACM/IEEE Conf. on Office Information Systems, Boston, MA, April 1990.

/Pro 89/ A.K. Proefrock, D. Tsichritzis, G. Mueller, M. Ader, "ITHACA: An integrated toolkit for Highly Advanced Computer Applications", in /Tsi 89/ and in Office and Business Systems Results and Progress of ESPRIT Projects in 1989, DG XIII, CEC, 1989.

/Pun 87/ P.P. Puncello, F. Pietri, P. Torrigiani, "ASPIS: a project on a knowledge-based environment for software development", CASE 87, 1987.

/Pun 88/ P.P. Puncello, P. Torrigiani, F. Pietri, R. Burlon, B. Cardile, M. Conti, "ASPIS: a knowledge based CASE environment", IEEE Software, March 1988.

/Pun 89/ W.W.Y. Pun, R.L. Winder, "A design method for object-oriented programming", in Proc. ECOOP'89, S. Cook ed., Cambridge University Press, 1989.

/San 90/ A. Sanfilippo, Dynamic INTRES, Graduation Thesis, Politecnico di Milano, 1990.

/Som 89/ I. Sommerville, Software Engineering, 3rd ed., Addison-Wesley, 1989.

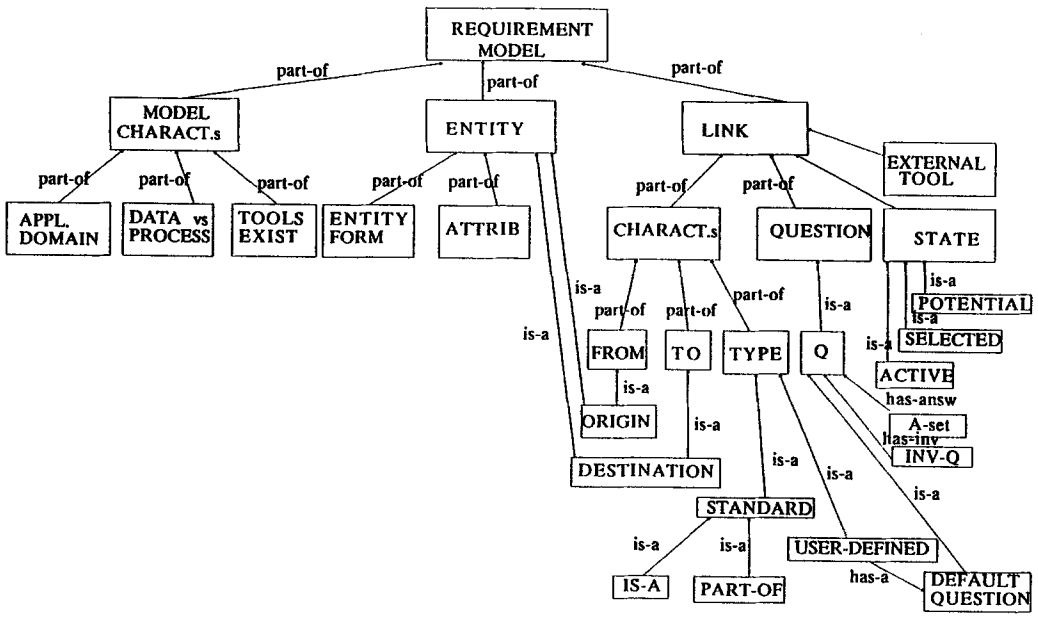
/Tsi 88/ D. Tsichritzis (Ed.), *Active Object Environments*, Centre Universitaire d'Informatique, University of Geneva, June 1988.

/Tsi 89/ D. Tsichritzis (Ed.), *Object-Oriented Development*, Centre Universitaire d'Informatique, University of Geneva, July 1989.

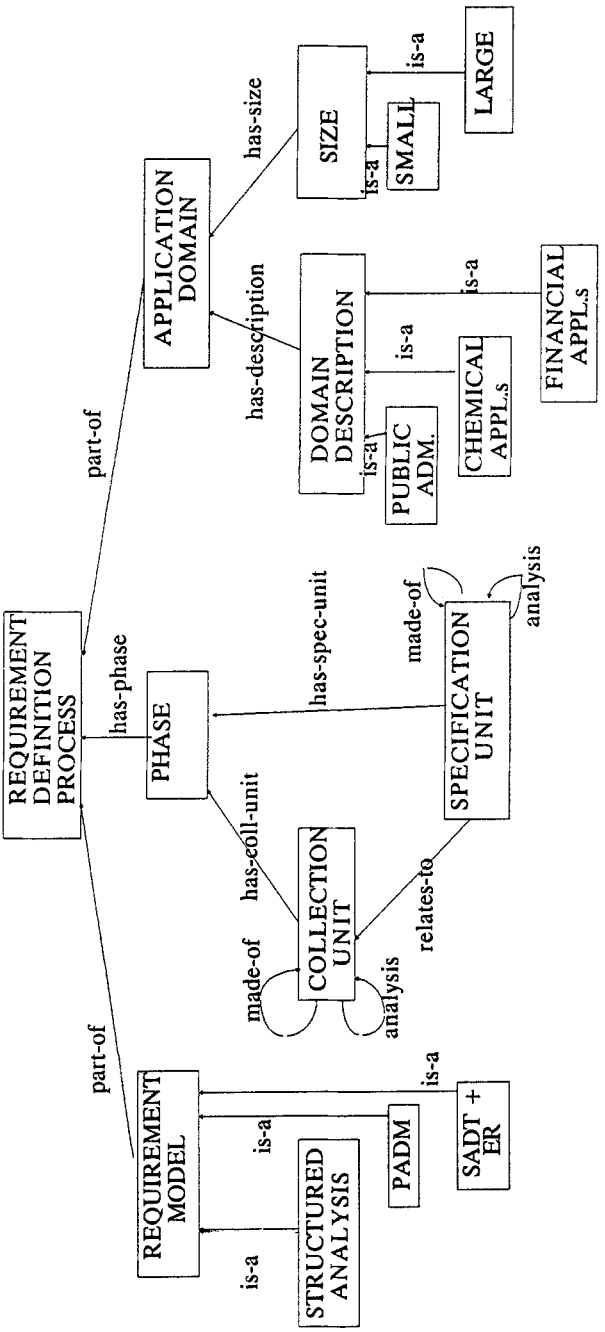
/Weg 87/ P. Wegner, "Varieties of reusability", in /Fre 87/.

/Wir 89/ M. Wirsing, R. Hennicker, R. Stabl, "MENU - An example for the systematic reuse of specifications", University of Passau *Technical Report MIP - 8930*, 1989.

Appendix 1 - Semantic network of RECAST meta--knowledge



Appendix 2 - Semantic network of RECAST knowledge



App. 2a

COLLECTION UNIT
part-of
ENTITY
is-a
CASE TYPE
has-a
SCHEMA OF STEPS
has-a
STEP
follows
has-a
AGENT
is-a
EXTERNAL OFFICE
is-a
OFFICE
has-a
DOCUMENT
is-a
OFFICIAL
is-a
APPROVAL
has-a
STATE
has-a
ACTION
has-a
SELECTION CONDITION
is-a
STEP DEFINITION
is-a
EXAMPLE OF STEP
is-a
ENTER EXAMPLE
is-a
TYPICAL DEFINITION
part-of
NON FUNCTIONAL
part-of
INTERFACE
part-of
SECURITY
part-of
ORGANIZATIONAL DIAGRAM
is-a
REQUEST
is-a
POLICE DOC
has-a
DEFINITION
is-a
EXAMPLE OF REQ
is-a
ENTER EXAMPLE
is-a
TYPICAL DEF

App. 2b

COLLECTION PHASE
next-phase
SPECIFICATION PHASE
has-a
SPECIFICATION UNIT
relates-to
COLLECTION UNIT
is-a
ORM OBJECT
part-of
NAME
part-of
ROLE
part-of
PROPERTY
part-of
OPERATION
part-of
RULE
is-a
REQUEST
has-design-suggestion
DATE
is-a
DESIGN OBJECT
is-a
OFFICE
has-design-suggestion
MAIL HANDLER
is-a

App. 2c