
# PUC

Monografias em Ciência da Computação
nº 21/92

# Establishing a Precise Relationship between a Software Development Environment and Software Process Models

Arndt von Staa
Carlos J. P. Lucena
Donald D. Cowan

Departamento de Informática

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO**

**RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453**

**RIO DE JANEIRO - BRASIL**

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Monografias em Ciência da Computação, Nº 21/92
Editor: Carlos J. P. Lucena Junho, 1992

# Establishing a Precise Relationship between a Software Development Environment and Software Process Models *

Arndt von Staa
Carlos J. P. Lucena
Donald D. Cowan #

# Computer Science Dept. and Computer Sistems Group at University of Waterloo, Ontario, Canada.

* This work has been sponsored by the Secretaria de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

**In charge of publications:**
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio — Departamento de Informática
Rua Marquês de São Vicente, 225 — Gávea
22453-900 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br
techrep@inf.puc-rio.br (for publications only)

# ESTABLISHING A PRECISE RELATIONSHIP BETWEEN A SOFTWARE DEVELOPMENT ENVIRONMENT AND SOFTWARE PROCESS MODELS

A. V. STAA C.J.P. LUCENA D.D. COWAN*

September 23, 1992

**Abstract**

Recent research has indicated that software development processes can be programmed or formally specified. Thus, it should be possible to build a software development meta environment or shell which can be adapted to specific process models. Building such a shell presents a technological challenge because the process programming languages and underlying database or repository must support multiple software process paradigms. This paper presents an outline of Talisman, one approach to establishing a relationship between the software process and the software environment. Talisman is a software development meta environment which can be adapted to several process models including the prototype, operational, transformational and successive refinement or "waterfall" paradigms.

Resumo

Pesquisas recentes demonstraram que processos de desenvolvimento de software podem ser programados ou especificados formalmente. Portanto, deve ser possível a construção de um meta ambiente de desenvonvimento de software ou um "shell" de ambiente que seja adaptável para modelos de processo específicos. A construção de um "shell" com essas características apresenta desafios tecnológicos porque a linguagem de programação de processos e a base de dados ou repositório subjacente ao ambiente devem permitir o suporte a multiplos paradigmas do processo de software. Este artigo apresenta uma visão geral do Talisman, um enfoque para o estabelecimento de uma relação precisa entre o processo de software e o ambiente de desenvolvimento. Talisman é um meta ambiente que pode ser adaptado a vários modelos de processo tais como o modelo de prototipação, o operacional, o transformacional e o modelo em fases.

**key-words**: Software Engineering Environments, Process Models, Software Development Repository, Process Knowledge Base, Objects.

**Palavras-chave**: Ambientes para Engenharia de Software, Modelos de Processo, Repositório do Projeto de Software, Base de Conhecimento sobre o Processo, Objetos.

*A.V. Staa and C.J.P. Lucena are with the Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Brazil. D.D. Cowan is with the Computer Science Department and Computer Systems Group at the University of Waterloo, Waterloo Ontario, Canada.

# 1 Introduction

It has been recognized over the last few years ([Dow87] and [ea88]) that software development environments need to be customized to fit the characteristics of different application areas, development teams, and technological and managerial cultures that exist in organizations. In order to customize a software development environment to support a particular project its architecture needs to be "open" and able to be configured for the many software process models which have been identified.

The different categories of software development processes are usually called software development paradigms [Agr86]. Whenever a particular process model is defined to fit a particular software project it is, most of the time, a specialization of one of the already identified development paradigms. Historically, the phased approach to software development, also called the successive refinement approach or the general "waterfall model" [Agr86] was the first development paradigm to be proposed when automatic support to software development was not a viable consideration. Other development situations motivated the prototype [Agr86], the operational [Agr86] and the transformational paradigms [Agr86]. The possibility of use of formal methods throughout the development process has recently suggested the contractual paradigm [Dow87].

A software process model can be used not only to define, but also to structure, direct, and record software development by having it enacted, partly by the developers and partly by the set of tools on which the model is implemented. A suitable software process model can provide a comprehensive framework for development, and enaction can provide traceability through the model of developer effort and its effects [ea92].

Recent research in software engineering has established ([Ost87, Wil88] and [Bjo87]) that software development processes can be viewed as computable objects. As such, they can be either programmed or formally specified. Thus, a relatively new research area has developed around the notion of establishing an appropriate relationship between software development environments and the software process. The design and implementation of languages or other formalisms for process formalization need to take into consideration several difficult problems [Not88]. For instance, process programming languages must support multiple paradigms, and environment repositories with properties beyond those currently available are needed to support the enaction of software processes. The difficulty arises in connection with the generality of the chosen formalism and the complexity it might impose on the structure of the environment's repository.

The design of a language or notation capable of representing equally well process models that adhere to most known development paradigms presents an interesting technological challenge. In this paper we describe the general approach used in the design of the Talisman software engineering environment to relate the environment tools to the description of process models in a way which makes it adaptable to a wide variety of software development situations.

Talisman is a meta environment or an environment shell that can be used by a software development team of a particular organization to develop a very wide class of software projects. Talisman uses a family of languages to model the software process: one different representation language to express each method in the adopted development approach and a form language to provide the semantics and other attributes of each element in a representation. An extended data dictionary associated with each representation used in a software project constitutes the Project Repository. The description of the relationships among the representations for a particular process model is stored in the Process Knowledge Base. Figure 1 illustrates this general overview of the Talisman

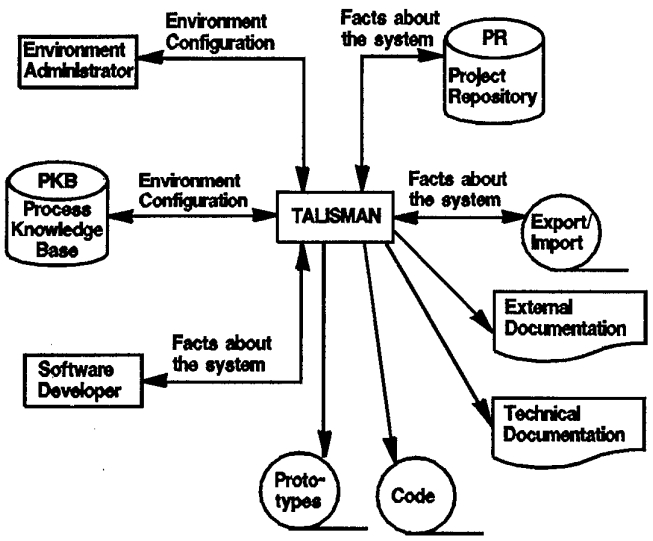


Figure 1: The Talisman meta environment

environment.

Different versions of Talisman have been used in industry and academia over the last 5 years. The version presented in this paper is used by 20 development groups in different companies to develop applications in the areas of office automation, technical engineering applications and information systems. Each development group works with different configurations of Talisman produced by themselves based on accumulated experience. As a starting point and to ease technology transfer problems Talisman provides an initial set of representation languages and a set of form programs that allow the description of process models that follow the general waterfall paradigm with method representations based on ideas from structured analysis and design as described in [Mar89, Mar90b] and [Mar90a].

# 2 An Overview of the Talisman Software Engineering Environment

Talisman is a meta environment which can be tailored to specific software development paradigms. This section provides an overview of Talisman by describing how different software development methods might be instantiated, the range of software activities which are supported and the form of the database or repository which supports the different environments.

## 2.1 Methods and their Representations

Following Wileden and Dowson's terminology [WD86], methods are assigned to the activities of a development approach for a given process model to instantiate how the approach can be placed in operation (for instance, the detailed design activity may be performed by a method that uses

pseudo-code as its representation). Different methods associated with different software development activities make use of particular software representations. The representations may incorporate diagrams or a variety of other kinds of "documents" such as data dictionaries, program schemata of various kinds, tables, and text fragments. Representations provide a "physical" visualization of the otherwise intangible software system under development.

The software development process inherently creates a large amount of redundant information. One of the original motivations for the work in Talisman was to provide a family of languages to allow the expression of an arbitrary set of representations of methods in a way that all redundant information that occurs in different phases of a development process can be either mechanically checked for consistency or generated automatically.

Talisman integrates the different representations associated with the different methods in a particular process model through a data base (data dictionary) that is called a Project Repository. Given the evolutionary nature of software development projects it is easy to appreciate that the motivation for this approach is related not only to the development process, but also to the various maintenance activities that take place during the software project life cycle. In fact, changes in one particular representation can be automatically propagated throughout the network of representations that document a software project.

Through export/import mechanisms it is possible to integrate the contents of various Project Repositories associated with the same global software development project, as well as to integrate the development of different projects. Integration via export/import mechanisms also allows for the establishment of corporate standards for software development.

## 2.2 Supporting the Different Functions Which Might be Present in a Software Development Team

The environment needs to support methods (tools) related to a very wide spectrum of activities including planning, specification, design, implementation, quality control, enforcement of standards, and management of the development process. Also the environment's tools need to be integrated, in the sense that the output from one tool is potentially the input to any other tool in the environment.

To achieve the integration between the environment and a process model, Talisman was designed with not only the development activities and their associated method representations in mind, but also the different functions that might be present in a generic software development team. The various members of the software development team and their relationships to Talisman are described next.

Customers and users demand additions or changes to an on-going project, and also approve the results. They must receive, in return, items such as an operational system, the technical documentation required to access the development of the project, and the user documentation. To enhance communication with the users Talisman supports the development of prototypes, the animation of specifications and the generation of technical documentation in various formats and degrees of detail.

The project manager defines the standards, notations and techniques to be used in a particular project. Specifically, the process model to be used including task structure and check-points, the standards for quality control and the various documents to be generated are defined.

The environment administrator configures each Talisman workstation. The configuration involves the implementation of quality standards and other standards and notations defined by the project manager. A configuration may cover the whole process or life cycle model or it can be confined to specific development activities. An environment administrator will normally prescribe one or more standard representation languages for each software developer, and may also define a new representation language together with its relationships to the other representation languages used in the project.

The data administrator defines the standards and data models that will be used for the corporation or for a particular project. The data administrator can establish the access rights and the standards for validation, data visualization and report presentation, thus, providing software developers with access to standardized data definitions in the Talisman data dictionary.

The data base administrator defines the access and manipulation rules for the data contained in the project's data bases. Talisman allows access to these rules and definitions as well as the possibility of their automatic incorporation into the code generated for the project.

Members of the software development group interact with Talisman via the representation language associated with their specific area of responsibility. Some of the elements contained in the specification might have been standardized. For instance, the data administrator may have defined data base tables and the data base administrator may have defined the physical realization of these tables in the available data base. The development group is responsible for design and coding. Transformers, programs and prototype generators may be available for most commonly used representation languages.

The maintenance team examines the Project Repository to become familiar with the project for later implementation of changes. Note that all documentation about the project will be represented in the Project Repository.

The quality auditors evaluate the quality of the different representations. Representation validators are programs written in the Talisman Form Language which is described in [SLC92]. These validations produce an audit report.

## 2.3 The Project Repository

The Project Repository stores objects and dictionaries. Each of these elements has several attributes. Additionally, a number of relationships may exist among the elements. Each object class corresponds to a linguistic element in one or more representation languages. Examples of object classes include: data store, process, entity, module, program block, task, data flow, and entity-relationship diagrams. For each object class a dictionary is defined. Dictionaries group together objects of the same class.

The object attributes can be grouped in several categories. The more important categories are:

**Names** uniquely identify an object.

**Aliases** are sequences of characters used as abbreviated names or elementary data (they are not part of the dictionary).

**Text fragments** are sequences of one or more lines of text used to express specifications, code, documentation and similar entities.

**Relationships** express links between two objects. Relationships are grouped in relations. Relations define permissions for the establishment of relationships between objects of two classes (same or different classes). The inter-related classes may belong to two different representation languages. If a relationship exists between a class of objects A and a class of objects B then there also exists an inverse relationship between B and A.

**Diagrams** are graphical representations. They contain object instances and labeled links. Objects may be related to diagrams through the relation "specified by" indicating that the diagram specifies a particular aspect of the object.

Individual diagrams are incomplete specifications. To complete them it is necessary to add more detailed specifications to each one of the element instances that appear in the diagram. In a simplified way we can say that diagrams are composed of the following elements: instances of objects, connections between instances of objects, labels to specify the nature of the relationships (they are also instances of objects), and terminations such as direction of flow in a data flow diagram to provide the details of the semantics of the relationships.

# 3 Talisman's Support to Process Models

Talisman supports a number of software process models including the prototype, operational, transformational, and phased analysis (waterfall) paradigms. This section provides some descriptive detail indicating how Talisman maintains the information for a specific process environment. The first part of this section describes Talisman's Process Knowledge Base that holds programs and objects which define a specific implementation of a process model. The remaining sections provide details about the various methods, objects and diagrams which are maintained in the Process Knowledge Base and their manipulation through various syntax-driven editors.

## 3.1 The Process Knowledge Base

Talisman was designed to allow the environment to be customized for the appropriate process model, development team, and type of development project including size and type of the application. The Process Knowledge Base defines the properties of a particular instantiation of a Talisman environment. Figure 2 illustrates the relationship between the Project Repository and the Process Knowledge Base in the Talisman environment. The ring around the Project Repository in Figure 2 is formed by the tools that implement the environment while the external ring is defined by the contents of the Process Knowledge Base.

The main definitions contained in the Process Knowledge Base are:

1. The representation languages available in a particular instance of the environment.

2. The set of classes of objects and corresponding attributes for each one of the representations available in the environment's instance.

3. The definition of the graphic languages: they specify the form of the external representation or user view, the internal representation or Project Repository storage mechanism, and the

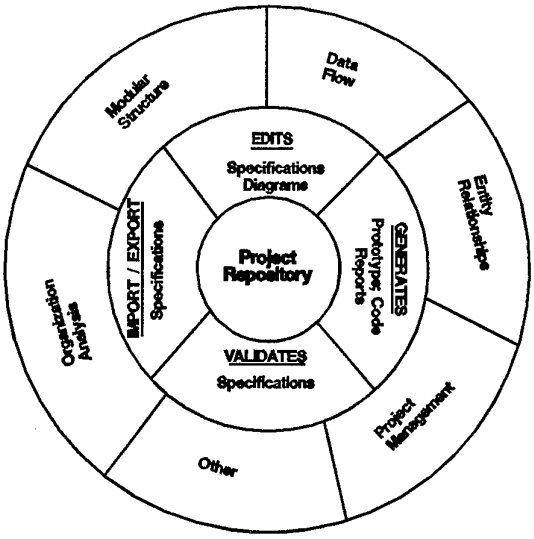


Figure 2: The relationship between the Project Repository and the Project Knowledge Base

editing procedure used for the representation or the method of interaction to create or alter the user view.

4. The form programs that express specifications: they define such objects as the attributes to be extracted, the screen appearance, and titles that constitute the specification text of an object from a given class. The specification form-programs determine the composition and the format of the specification for each one of the objects that compose the software project which is being developed. These programs also allow code fragments or documentation text to be associated with the objects.

5. Form-programs for validation: they define the rules for syntactic and semantic checks of the content of a given object or diagram. The application of a validation form-program to a given object generates a "validation report" about the object. The report will contain all errors detected by the validation procedure. The validation-form programs guarantee the semantic validity of the Project Repository.

6. Form programs for linearization and export purposes define the search and diagram structuring processes developed with the information registered in the Project Repository. These form programs aim at the generation of files containing entities such as source code, formatted text, and data declarations for interface layouts text to be exported. Form programs for linearization may also be used for the construction of program prototypes.

7. Form programs for output preparation define entities such as the attributes to be extracted, report layouts, and titles that compose the reports to be produced from the objects of a given class.

8. Form programs for transformation define how a given representation can be transformed into another one.

9. All parameters that govern the operation of the present environment instance, such as command languages, dialog and menu definitions.

## 3.2 Representations of Methods

Most current specification and design methods make use of graphical or visual representations. Some popular visual representations are: data flow diagrams, entity-relationship diagrams, state and transition diagrams. The diagrams that correspond to each of these notations can be expressed by means of a special purpose representation language. For example, a data flow diagram is expressed in a data flow representation language.

Other methods that can be associated with the software development process make use of "forms" that allow the application of cross-reference checks against the results of processes such as requirements or organization analysis. Each form is a representation and each composite form, produced by means of a linearization process which navigates through the cross-reference links established among various forms, is also a representation.

Diagrams of any kind are considered incomplete representations of specifications associated with a method such as the recording of software requirements or software design. To have a complete specification the meaning of all language elements used in the visual representation such as different boxes, and arrows must be defined in detail.

The specification of these elements is expressed in Talisman by a form program. For each class of element such as process, data structure, data store, entity, or relationship, a form program must be defined. Each form program contains various fields which stand for: titles, aliases, text fragments, relations and other system elements. The form fields are attributes associated with the elements.

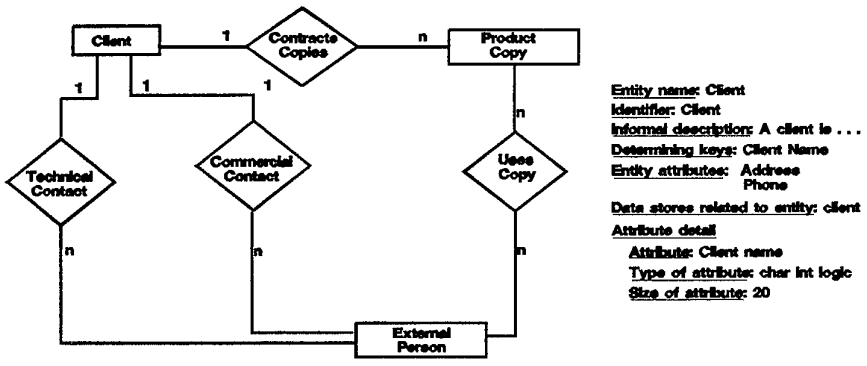


Figure 3: A sample form program

In Figure 3 we illustrate the pair (diagram, form) used in an entity-relationship representation language. In the form the fields are shown underlined while the normal text is provided by the user of the form language.

### 3.2.1 Relating Different Representations

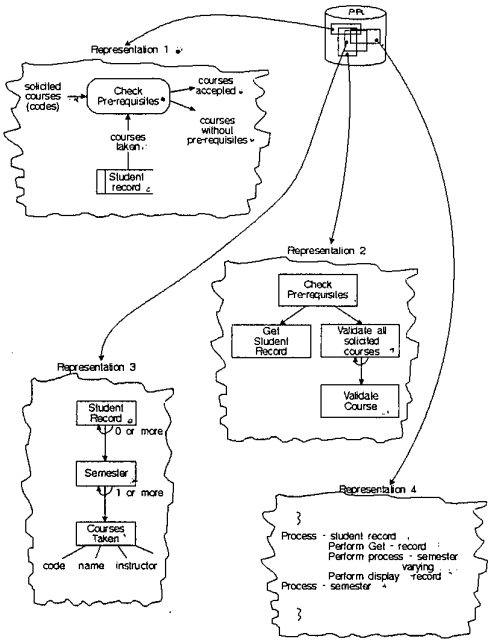


Figure 4: Several representations or views of a development project

The Project Repository contains all the recorded information related to a particular software development project or set of projects. In the repository, each representation can seen as a view, in the data base sense, of the information about a development project. Figure 4 illustrates this concept. The different views (representations) overlap. This desirable feature of the environment design is also responsible for the inevitable complexity of the task of the environment administrator.

As we have seen, each representation is expressed in a representation language. Each representation language has a specific syntax and semantics and is, in general, formed by a combination of diagrams and forms. The design of a representation language to support methods associated with the various activities of the development process usually has to address the following three problems: levels of abstraction, coordination of views, and level of formality. The language needs to capture the features usually incorporated in representation languages to express different levels of abstraction. For example, data flows can be decomposed into lower level data flows. Also different representation languages can be used to capture complementary aspects of many items such as a requirements specification. It is common to use the data flow approach to represent the functionality of a software system and an entity-relationship diagram to express the data dependencies of the same system. The way the two representations interact should be expressed in a form program. The form language offers considerable flexibility. It is possible, for instance, to define a formal semantics for the elements that compose a requirements diagram. Nevertheless, this linguistic level would be entirely inappropriate to communicate with the customer that is supposed to validate the requirements representation.

Representations are not static sets of information stored in the Project Repository. Representations should allow for the transformation of the information it incorporates. Several types of operations can be applied to a representation in the Talisman environment. We can refine or add detail to a representation, we can compose the information contained in two representations by reflecting one into the other and perform other similar operations.

The concept of representation transformation allows considerable flexibility in the expression of a variety of process models based on different software development paradigms. If we can refine a representation associated with an early phase of the development process by adding sufficient detail that allows us to produce a prototype, we would have a process model that follows the prototype paradigm [Agr86]. If we adopt representations of abstract machines and define how to transform one machine into the other we would follow the operational paradigm [Agr86]. If we define a sequence of representation languages each associated with more detailed activities of the development process such as specification, design and implementation, we would follow a phased development paradigm like the waterfall approach [Agr86].

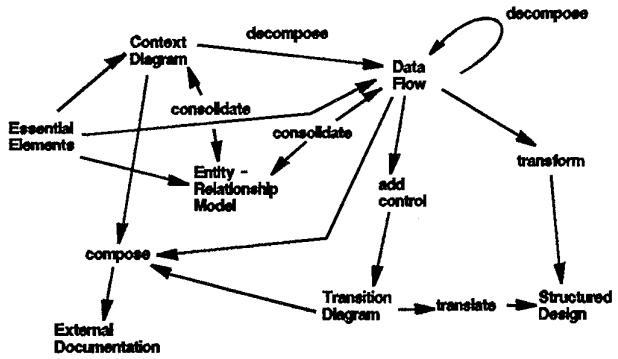


Figure 5: A network of representations

During development and maintenance, representations are systematically modified. Modifications are meant to correct or enhance representations. After modifying a representation it is necessary to propagate the changes to the other representations. If the representations are ordered according to a given process model it may be necessary to backtrack to reflect modifications. For instance, a change in the source code representation may reflect back to the requirements representation. The set of representations form a network as shown in Figure 5. In the Figure the nodes stand for different representations and the edges represent the existing transformations in a process model. This model is based on a current version of structured analysis and design which incorporates the so called "essential analysis" [Mar89], and in the use of data flow and entity-relationship diagrams to express different views of the requirements.

In Talisman it is possible to transform representations using form programs. It is also possible to relate elements of different representation languages using the form editor. For instance, it is possible to associate program blocks (structured design) to processes (data flow). This way the information from the data flow may be used to generate the code as shown in Figure 6.

Talisman allows the browsing of representation networks in a way that resembles hypertext navigation through text fragments and interrelated diagrams. The cross references as shown in

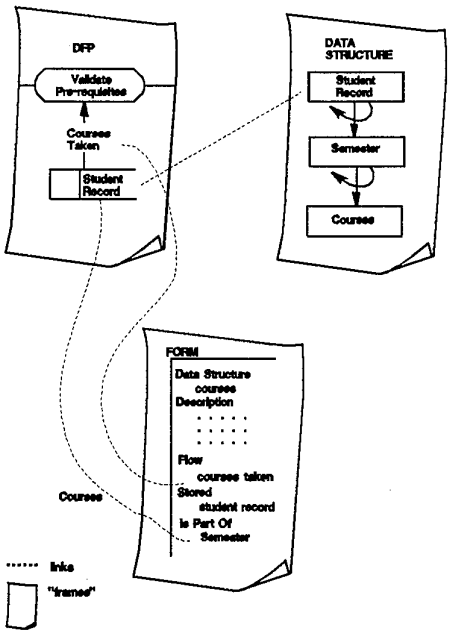


Figure 6: Code generation in Talisman

Figure 6 that relate the representations, establish links among parts of the representations or frames. Navigation may take place under the control of a software developer through the use of navigation commands. It may also be programmed as form programs. These programs allow the manipulation of information in the Project Repository expressed as objects and attributes.

### 3.2.2 Objects and Diagrams

Talisman integrates the representations by using a given class of objects in different representation languages or by relating the elements that compose the various representations. These relations allow functions such as validation, report generation and code generation.

Each object in Talisman belongs to a class of objects. Different representation languages can share classes of objects. For instance, the class of attributes of the representation language entity-relationship is the data structure class in the data flow language. The class processes is used by the data flow, state and transition, and project management representation languages.

For each class of objects a default specification form is defined. The forms are provided to support the standard representations provided by the environment. These forms may be re-defined or new forms may be developed using the form language. The forms contain text and relations fields. Not all fields will be filled in all forms.

Objects are externally identified by their names or their associated aliases. Diagrams in Talisman are essentially graphs formed by instances of objects and labeled links. Each object is specified only once in the Project Repository. The specification is accessible to all possible representations via their respective data dictionaries. Processes, states and entities are examples of objects.

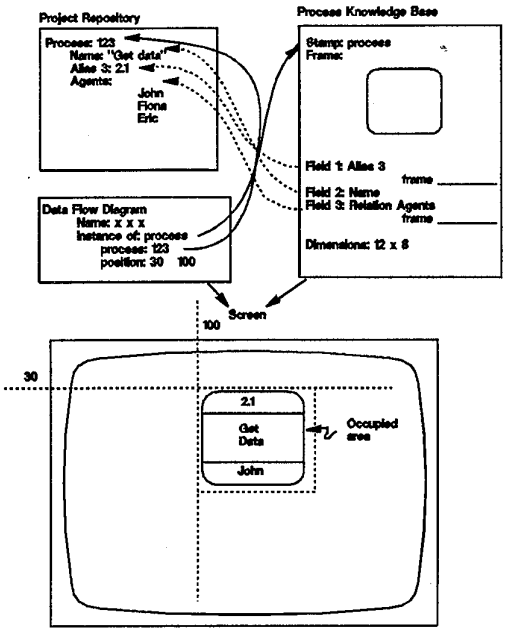


Figure 7: The composition of an object instance

When an object instance is created, the object stamp in the Process Knowledge Base and the object specification are put together to form a visible instance in a diagram. The values to be searched in the specification are defined in the stamp. Figure 7 illustrates this composition. The instantiation generates cross references, that is, it allows Talisman to determine all places in which objects are used. This feature is essential for the purposes of quality control and project management. In other words, all changes to a given object occur only once.

Links establish relationships or interfaces between object instances. The label and the orientation of the link define the nature of this interface. A link may be undirected uni-directional or bi-directional. Each link may be unlabelled or have a single label. Each representation language determines the class of objects which can be part of a label. In the data flow representation language the labels belong to the class data flows. In the entity-relationship representation language the labels belong to the class process. A label is similar to an object instance. Both an object specification and a label specification are defined by a specification form. A label's name cannot be empty but a link may not have a label.

### 3.2.3 Syntax Driven Editors

Talisman makes use of several syntax driven editors which are oriented towards specific tasks. Each editor is instantiated by means of a definition contained in the Process Knowledge Base.

The dictionaries editor is an access mechanism to objects contained in the Project Repository. It allows for the insertion, exclusion and change (via another editor) of the objects contained in the Project Repository. Access to all editors is gained through the dictionaries editor.

The form editor edits the attributes of one or more objects searched and displayed by means of a form program. Intuitively the form editor may viewed as an access language to the Project Repository which accesses, formats, displays and edits data. The form editor also allows the navigation between different specifications (forms) in a hypertext style.

The diagram editor supports the creation and change of arbitrary diagrams. The Process Knowledge Base instantiates the diagrams editor for each one of the available graphical representation languages. Each object instance displayed in a diagram has a specification in the corresponding dictionary. Access to the specification can be made directly just by pointing to the instance and soliciting an edition. In general, the diagram editor allows the direct manipulation of all elements displayed.

The structure editor edits object structures from the same object class. Structures usually form object hierarchies. By means of the structure editor it is possible to edit program modules defined by a particular sub-set of object classes and relations between those classes. The structure editor may be used to edit a variety of organizations. For instance, it can be used to edit program modules, data structures, documents and task or work break-down structures. Each of these structures is defined through a hierarchy of blocks. Each block has several attributes. These attributes may be created and edited via the form editor, which may be directly activated from a given block. The structures may be converted to become sequential files which may be used as inputs for processors such as a compiler, or a formatter.

# 4 Quality Control With Talisman

Talisman supports various kinds of quality control activities. A software development process supported by the environment may assure quality:

1. By construction, since all interdependent information is adjusted when any one of them is edited. For instance, when a name is edited all places where this name is used are automatically updated.

2. By the use of protection mechanisms, modifications which are not allowed by definition in a given representation language are automatically inhibited when diagrams and/or forms are used. For instance, the data flow diagram editor does not allow links between external entities and data stores.

3. By validation procedures, validation forms are applied to the different elements that compose a specification and an audit report is issued with the errors detected. For instance, it is possible to produce validation forms which are able to check if the data that flow to a given data store can legally be stored in it.

4. By inspection, expert judgement is applied to the representations. The definition criteria and the results expected from the evaluation may be defined in a form (the evaluation result can be composed with the audit report through validation forms).

The validation of the Project Repository produces audit reports. These reports together with the demands for change, coordinate the change and process of change propagation. Audit reports

are text fragments generated by the validators. In Talisman it is possible to develop validation programs in the form language to establish various levels of quality control. Usually, the texts of audit reports are protected in such a way that the developer cannot change them unless new validation programs are written.

In addition to automatic validation procedures, most of quality control can be informally performed by experts. The informal control is mostly performed through walkthroughs. To support walkthroughs, Talisman allows the exploration of the Project Base through hypertext mechanisms.

# 5 Integrating the Talisman Environment With Different Process Models

It has been claimed that Talisman supports several different process models. This section describes how the Talisman meta-environment is used in support of the prototype, operational, transformational, and phased analysis (waterfall) paradigms.

## 5.1 The Prototype, the Operational and the Transformational Paradigms

It is possible to produce compilable code with Talisman. It can be done in the following two ways:

1. linearization: the organization of the content of the Project Repository in a sequential form;
2. generation: program composition from a schema completed with information contained in the Project Repository.

Generators may allow the elimination of various development steps used, for instance, in the traditional phased approach to software development. Generators are dependent on particular applications and must be written in the form programming language. Generators can also be used to produce prototypes. Prototyping in Talisman offers the additional advantage that all information used to produce the prototype will be later available for the development of the final software product or for the continuation of the prototyping process.

Program generation is obtained by combining solution schemata with specifications in a process that resembles a macro expansion. Schemata are program skeletons (designs or programs-in-the-large [DK76]) together with rules for completing them. Schemata are instantiated during the generation process through the selective addition of information from the specification. The same technique may be used to transform one representation into another.

The Talisman features just described indicate how process models based on the prototype, operational and transformational paradigms may be instantiated with the environment's support. The environment should be configured through the definition of the appropriate representation languages and associated form programs. To implement the operational approach [Zav84] in Talisman the environment administrator would need to specify the objects and their respective properties for the abstract machines and define the set of transformations among the machines. The developers using the described process model would also like to develop generators for the abstract machines in the model. Even though the code generated for the higher level machines is not as good as

the code produced for the lower level ones, they would serve the same purpose as prototypes in a prototype-based process-model.

Note that since the representations and respective transformations will be stored in the Process Knowledge Base, the heuristics for the selection of transformations, if they are known, could also be recorded as form programs thus providing the particular configuration of Talisman with so-called knowledge-based characteristics.

## 5.2 The Phased Approach to Software Development and the Methods of Structured Analysis and Design

Talisman adopts a default process model based on the classical phased approach to software development. All steps in the so-called waterfall model will be covered and this process is independent of the order in which the development activities occur. Tasks may be distributed to the members of the development team to take advantage of parallelism whenever this is possible. Simulation and prototyping can also be used to supplement the process.

To support the direct application of its default process model, Talisman is originally configured with the following representation languages: a language for project identification, a language to express the analysis of organizations, a requirements analysis language, a language for project management, a language for entity-relationship diagrams, a data flow language, a state and transition language, a modular structures language and a documentation language.

Besides the standard representation languages and form language programs supplied by Talisman for the process model, a number of other standard procedures can be used by the development team. For instance, a corporate data dictionary can be used to support the generation of specifications. Through this procedure, when a data model is created for the project much of its data will be already available including manipulation operations and presentation formats.

In what follows we provide a short description of the environment support for the default process model.

1. The conceptual phase is used to register the rationale and purpose for a development project; it is usually a short text in natural language.

2. Domain analysis involves the identification of the generic elements that will participate in all similar applications; the identification of the generic elements and the definition of their interrelationships allows standardization of the solution for a given class of problems or components of applications.

3. Process analysis includes the identification of development processes and their managerial aspects. This activity will be supported in the default Talisman environment by a representation language for organizational analysis; the supported activity allows for the identification of the organization areas involved in the application, the managerial processes and the functions of the specialists that will be involved with the use of the project.

4. Requirements analysis is supported by the representation language for requirements analysis.

5. Essential analysis:

(a) The static specification defines the data structures to be used such as tables, lists and data base files. The activity is supported by the entity-relationship representation language.

(b) The functional specification defines how a valid state (its input data satisfy given pre-conditions) is transformed into another state (its outputs satisfy given post-conditions); the specification is said to be functional because it defines the results to be produced from process (function) input data. It also defines the composition of functions. To define the functional specification the default Talisman configuration provides the data flow representation language.

(c) The dynamic specification defines the sequence of process execution including parallel execution. The representation language used for this activity is the state and transition language.

The representations in the languages are implicitly integrated through the use of the same class of objects in various languages. For instance, the attributes of an entity in the entity-relationship representation language are the same class of objects as the data structures in the data flow language. The processes that occur in the data flow representation belong to the same class of objects as the processes referred to in the state and transition representation. Additionally, the representations can be connected through explicit relationships among the various objects.

6. General Architecture: the programs to be developed are identified, computational resources are assigned to the programs, multiprocessing and resource sharing situations are identified and the interfaces between modules are defined. This information is added to the specifications by using the representation languages in which they have been expressed. The development process is also detailed through the definition of the products to be generated in association with the various activities. All this information is recorded by means of the project management representation language.

7. Program Modularization (detailed design): in the default Talisman process model, the form of structured analysis described earlier is transformed into a structured design. The activity involves the precise definition of the modules to be developed or reused. This activity is supported by the modular structure representation language.

8. Structured Design and Coding: this activity involves the internal organization of the modules to produce structures annotated with code that will allow the direct generation of compilable code. The same modular structure representation language is used in this case.

# 6 Conclusions

Talisman is a meta environment or shell for building software development environments tailored to a specific process model. The Talisman design (representation languages, form language, Project Repository and Process Knowledge Base) allows a consistent integration between the environment shell and software process models. The redundancy inherent in any software construction process

is supported since only one copy of each object is kept in the Project Repository, but appropriate links ensure that consistency is maintained.

Talisman has been used successfully in a substantial number of software projects ranging over a number of different application areas. This experience provides a strong indication that the approach supported by Talisman is viable. Based on the experience collected from these and other experimental projects, we are making further refinements to the Talisman model.

## References


[Agr86] William W. Agresti. *New Paradigms for Software Development: Tutorial.* IEEE Computer Society Press, Computer Sciences Corporation, Silver Spring, MD, 1986.

[Bjo87] D. Bjorner. On the Use of Formal Methods in Software Development. In *Proceedings of the Ninth International Conference on Software Engineering*, 1987.

[DK76] F. DeRemer and H. Kron. Programming-in-the-large versus programming-in-the-small. *IEEE Transactions on Software Engineering*, 2(2), 1976.

[Dow87] M. Dowson. ISTAR - An Integrated Project Support Environment, Proc. 2nd ACM SIGPLAN/SIGSOFT Software Eng. Symposium on Practical Development Support Environment. *ACM SIGPLAN Notices*, 2(1), 1987.

[ea88] R. Taylor et al. Foundations for the Arcadia Environment Architecture. In *Proceedings of the [third] ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, November 1988.

[ea92] T. Shepard et al. A Visual Software Process Language. *CACM*, 35(4), 1992.

[Mar89] J. Martin. *Information Engineering - Introduction.* Prentice-Hall, 1989.

[Mar90a] J. Martin. *Information Engineering - Design and Construction.* Prentice-Hall, 1990.

[Mar90b] J. Martin. *Information Engineering - Planning and Analysis.* Prentice-Hall, 1990.

[Not88] D. Notkin. The Relationship Between Software Development Environments and the Software Process. *SIGSOFT Software Engineering Notes*, 13(5), November 1988.

[Ost87] L. Osterweil. Software Processes are Software Too. In *Proceedings of the Ninth International Conference on Software Engineering*, 1987.

[SLC92] A. v. Staa, C. J. P. Lucena, and D. D. Cowan. TALISMAN: A Process Model Driven Software Engineering Environment. Technical Report CS-92-36, Computer Science Dept. University of Waterloo, Waterloo, Ontario, Canada, June 1992.

[WD86] Jack C. Wileden and Mark Dowson, editors. *International Workshop on the Software Process and Software Environments.* ACM SIGSOFT, ACM Press, August 1986.

[Wil88] L. G. Williams. Software Process Modeling: A Behavioral Approach. In *Proceedings of the International Conference on Software Engineering*, 1988.

[Zav84] P. Zave. The Operational Versus The Conventional Approach to Software Development. *CACM*, 27(4), 1984.