Database Support for Cooperative Work Documentation

Agnès Voisard
voisard@inf.fu-berlin.de
Computer Science Institute
Free University, Berlin, Germany

Claudia Bauzer Medeiros
cmbm@dcc.unicamp.br
IC - UNICAMP - CP 6176
13081-970 Campinas SP Brazil

Geneviève Jomier
jomier@lamsade.dauphine.fr
LAMSADE - Université Paris IX
75775 Paris CEDEX France

Abstract. Technological changes impose a constant evolution on all kinds of artifacts, and require new solutions for their efficient maintenance. Appropriate documentation is considered fundamental for maintenance and evolution. This situation is even more crucial when one considers today's cooperative environments for designing and developing artifacts. Most of the time, documentation is static and describes WHAT an artifact is, and sometimes HOW it was designed and constructed. Moreover, in collaborative work, documentation serves as one of the communication means among all involved in creating an artifact. However, several other types of documentation needs have been identified in many domains – e.g., medicine, engineering, biology or astronomy – such as flexible versioning for keeping track of an artifact's entire evolution, as well as documentation for the reasoning (the WHY) behind its construction. Unfortunately, no comprehensive system exists to handle all these documentation requirements: each kind of document is managed by a separate system, and furthermore studied in a different Computer Science field. WHAT documentation may fall within database or software engineering research, whereas HOW is often restricted to hypermedia systems and CSCW, and WHY is handled in the context of Artificial Intelligence and cognitive science.

This paper presents a unified framework to manage all these kinds of documents within a single database, for engineering artifacts. This allows integrating and coordinating the (cooperative) work of different types of users of these artifacts: designers, customers, salespeople, constructors. This eliminates the break in continuity found in normal environments, where each kind of documentation is handled separately and uses distinct implementation paradigms. Our framework is exemplified in the context of software module configuration.

1 Introduction

An artifact is "something created by humans for a practical purpose" (Webster's Dictionary). Engineering artifacts are those which are built "using a combination of different sciences and mathematics applied to distinct materials and sources of energy". Canonical examples of these artifacts are buildings, cars or telecommunication networks. Other less commonly cited engineering artifacts include new vegetable seeds (in genetic engineering), maps (in geography/geology) and software (software engineering).

The construction of complex artifacts frequently involves several groups of experts, working in teams, often geographically distributed. These teams or individuals must cooperate during different work stages in order to design and construct the artifact (building, map, seed, software). The more complex an artifact and the technology involved, the greater will be the amount of people participating in its production and the higher the costs for its maintenance and upgrade.

Normally, artifacts will evolve through time (either through improvements in design or construction techniques, or following changes in requirements/specifications, or even through use and decay). The process of creating one artifact may never end, configuring a situation in which several artifact *versions* may co-exist. Traditionally, two kinds of versioning are considered: versioning resulting from evolution in time (so-called linear or historic versioning); and versioning due to design/implementation alternatives (so-called arborescent or parallel versioning).

The maintainability and evolution of an artifact are highly dependent on the associated documentation. The more complex an artifact, the greater the number of people and technologies involved in its construction and therefore the bigger the need for documentation. The term "documentation" covers a broad spectrum. In the engineering context, documents are associated with data generated during the design and construction of an artifact. In particular, these documents keep track of three different issues:

1. Data describing the artifact (e.g., manuals, metadata, text usually organized in hypertext structures) – here called WHAT documents

2. Data describing the process used to arrive at the end-product (e.g., mathematical models, procedures, workflows) – here called HOW documents

3. Data describing the rationale behind the design of the artifact – here called WHY documents. This is a new research area, called *design rationale* [**?**]. A novel aspect of our work is that we consider rationale as applied to all stages of an artifact life cycle.

Solutions have been proposed to all these issues, but so far they have been handled separately with each kind of document managed by a different system. It is up to the teams involved in the design and construction of an artifact to manage the relationships among artifact versions, and those connecting artifacts and documents.

This paper presents WHOW – a database-centered framework to solve this problem. The DBMS stores artifacts and WHAT, WHY, HOW documents. The framework relies on a general versioning mechanism – the DBV mechanism [**?**], which handles uniformly all kinds of versioning (schema, extensions, constraints, configurations), thereby allowing users to propose alternatives to a product by trying out distinct reasonings and to keep track of links among artifacts, configurations and the associated documentation. This unification simplifies the issues of documentation, cooperative work and evolution support. Users working in a cooperative environment will manage their documentation through this framework, which will integrate them in a coherent manner, decreasing the problems encountered in distributed design and development environments.

The rest of this paper is organized as follows. Section **??** presents related work on documentation. Section **??** presents why-graphs, which is the formalism proposed to express and store the rationale for an artifact's life cycle (not just its design stage). The architecture of the WHOW framework, together with the the specific needs it answers, is presented in Section **??**. Section **??** shows how this framework can be applied in a specific domain – management of software module configurations. Section **??** presents conclusions and ongoing work.

2 Related work

Related work concerns discussion on the three kinds of documentation needed for artifacts. So far, to the best of our knowledge, no work in the literature combines these document types. We center on artifact documentation issues, regardless of the nature of work involved – whether used in a cooperative environment or not, these types of documents are considered as necessary at some point in an artifact's life cycle.

The term "document", when associated to an engineering artifact, in a Computer Science context, refers most of the time to static documents (i.e., blueprints, specification requirements, or manuals). Considered from a database point of view, artifact documents are stored as

◇ Metadata – special kinds of records that describe the artifacts from a more abstract point of view. Their goal is to help organize, identify and retrieve specific artifacts. Metadata are also found in collaborative work contexts, as a means to "index" useful data across user workspaces.

◇ Textual data, organized in hypertext/hypermedia graphs – these documents describe specific characteristics of the artifacts (e.g., requirements, manuals, specification diagrams). Their goal is to ensure maintainability and usability of the artifact.

◇ Workflows – a more recent type of (dynamic) document associated to artifacts, workflows register the process used to build the artifact. This new form of documentation is usually employed business activities where several agents may be involved. It is also found in scientific environments, when artifacts are the result of scientific experiments, and the workflows (the so-called *scientific workflows*) describe the steps and algorithms which were used to construct the artifact. Their goal is to ensure maintainability (by describing the process) and repeatability in reproducing the artifact.

The use of textual data and hypermedia facilities for documentation purposes is already well established, and needs not be extended here, since we do not propose any novelty in this sense.

Metadata usually describe the genealogy of an artifact (who created it, and sources of its creation), validity information and quality indicators. Geographical metadata are concerned with "where" issues, whereas biological metadata worry about taxonomic problems. Metadata for engineering artifacts, from a higher level perspective, concerns authoring, schedules and temporal validity, and sometimes the procedures used to create an artifact. For discussions on different types of metadata standards, the reader is directed to [**?**, **?**].

A *workflow* denotes the controlled execution of multiple tasks in an environment of distributed processing elements [**?**]. It can be defined a set of tasks involved in a procedure along with their interdependencies, inputs and outputs. Workflows are gaining increasing acceptance as a means of documenting and organizing procedures, as well as helping the coordination of groups. The term *scientific workflow* [**?**] was coined to denote a specific kind of workflow which can be used to control the execution of scientific experiments and procedures. Workflows for documentation purposes - and, more specifically, scientific workflows - are discussed, for instance, in [?, ?, ?, ?].

A new type of documentation which is still object of research in AI and the cognitive sciences is the so-called *design rationale*, which is being used experimentally in the contexts of software engineering and human-computer interfaces. *Design rationale* represents the reasons and reasoning process behind the design and specification of artifacts [?]. It allows keeping track of assumptions made during the design process, and the discussions conducted within a design team – and sometimes across teams – to arrive at a given solution.

A novel aspect of our framework is the way in which we integrate this type of documentation to the other kinds of documents. Thus, this related work section is mostly devoted to rationale. Rationale is a WHY kind of document, as opposed to WHAT (metadata and texts) and HOW (workflows). As pointed out in [?], documenting rationale is important because an artifact needs to be understood by a wide variety of people who have to deal with it - from designers and users to instructors, salespeople and maintainers. To these users, it is not only the artifact that is important, but also other issues - how to change it, how to market it, and so on, since there are many ways of working with the artifact.

Among the several flavors of design rationale defined in [?], we are especially interested in two: (i) an expression of the relationships between a designed artifact, its purpose, the designer's conceptualization and the contextual constraints on realizing the purpose; and (ii) documentation of the reasons for the design of an artifact, the stages or steps of the design process, and the history of the design and its context.

Formal methods for documenting rationale are usually based on directed graphs, where edges and nodes acquire specific semantic meaning. Since these graph structures become unmanageable as nodes and links grow, hypertext tools are often used in conjunction with these representations. Examples are the Design Space Analysis proposal of [?] and the IBIS/gIBIS system of [?]. Both use directed graphs as semi-formal methods to describe discussions and dialogues in collaborative design deliberations (gIBIS) and questions, options and criteria considered during the design activity (Design Space Analysis).

IBIS (Issue-Based Information System) seeks to capture the issues that arise in the course of design deliberation, along with the various positions (or alternatives) that are raised in response to issues, and the arguments for and against the positions. Its graphical hypertext tool – gIBIS [?] – represents this argumentation as a tree, rooted at the issue where the conversation among designers start. This tree is based on three main concepts: Issues, which are presented for discussion (the root is the main Issue); Positions, which respond to the issue with one or more tentative solutions; and Arguments, which support or object to these positions. The tree grows asb secondary issues (and positions and arguments for them) appear to question or expand the discussion on the root issue.

Design Space Analysis [?] is a methodology to construct the documentation of the design rationale for an artifact. It is based on three elements – Questions, Options and Criteria. Its main constituents are Questions which are posed on design rules, Options, which are alternative answers to these questions, and Criteria for assessing options. Q, O, C are linked in graphs. Criteria organize the reasoning, giving priority to some options over others.

Related research concerns means of extracting the rationale from designers and documenting it in an appropriate way. Finally, further research on rationale covers cognitive and psychological aspects. All these issues are outside the scope of this paper.

The next section describes the structure adopted to document the rationale behind the design and construction of an artifact – why-graph. We assume that some type of methodology has been applied to extract the relevant information from designers, and store it into this structure.

3 Supporting rationale documentation in why-graphs

This section presents why-graphs (WG), which is a semi-formal method for representing rationale. Unlike other rationale formalisms, why-graphs are stored in a database. This means not only that they can be managed together with the artifacts they explain, but also that arguments for a given artifact can be reused to construct other artifacts. They constitute one of the contributions of this paper, since they extend the concept of *rationale* (at present restricted to design) to encompass the entire life cycle of an artifact – design, implementation, construction, testing and maintenance. Indeed, whereas existing rationale formalisms are restricted to why a given object was designed in a certain way, why-graphs also document why it was constructed, maintained and tested, i.e., why it *is* the way it is.

Definition 3.1. A *why-explanation graph*, or why-graph (WG) for short, is a pair (E, W), where

- E, the set of nodes, are explanations. An explanation is any kind of entity or argument which a user sees fit to use as part of a rationale construction. Examples of explanations are a text, a formula, a theory, a theorem, a photo. Explanations may be simple or complex.

- W is a set of edges. There is an edge e from a node n_i to a node n_{i+1} if n_{i+1} is an explanation of n_i. The semantics of an edge e between n_i and n_{i+1} is "n_{i+1} *is an explanation of* n_i".

Definition 3.2 An *explanation chain EC* is a pair (A, P) where A is a (documented) artifact and P a path ($P \subset WG$, where WG is a why-explanation graph).

An example of an explanation chain in a why-graph is "Theory t is used to construct justification argument j". t and j are nodes (j is a complex explanation), linked by the edge "used-to-construct". Justification j explains WHY a given artifact was constructed a certain way.

Note that an explanation is not a simple textual expression. It is itself a complex object based on other explanations, which materializes the thinking process. Under this perspective, the graphs used in the literature to formalize design rationale (e.g., IBIS graphs [**?**]) can be considered as a special case of why-graphs – for instance, justification argument j above may be typically a design deliberation session documented in IBIS. Moreover, since a design rationale document *explains* a given stage (design phase) of artifact construction, an entire IBIS graph can be treated as a complex explanation E and thus a node of a why-graph.

An artifact A is associated with one or several explanation chains that explain the WHY behind it. If several explanation chains exist for an artifact, this indicates that distinct acceptable reasonings can be applied to it.

These reasonings can represent WHY-alternatives to an artifact or may be complementary reasonings provided by different groups that participated/collaborated in the construction of the artifact. For instance, given a piece of software, some of its characteristics may be due to performance requirements (technical reasons) whereas others may correspond to customer cultural needs (marketing, social reasons). Section **??** gives an example of an artifact with multiple explanation chains in the context of software module configuration.

Definition 3.3 An *interpretation* I_A of an artifact A is a set of explanation chains.

Intuitively, a given interpretation of an artifact chooses how to explain it in case several rationale are associated with the artifact. Multiple interpretations can be defined by extracting paths in the why-graph.
Thus, an artifact can be associated with various interpretations.

Operations on why-graphs are traditional graph operations. One can therefore document and construct complex rationale by combining parts of why-graphs; alternatively, one can prune these graphs, eliminating some non-verified explanation or explanation chain. An additional operation is the one that changes the content of a node – e.g., changes an explanation.

4 Documentation integration in WHOW

This section describes the WHOW framework as a platform for integrating the various kinds of WHAT, WHY and HOW documentation. We start with illustration examples borrowed from a wide variety of applications, and then focus on the WHOW type of documentation.

4.1 Illustration: Reference queries

Our framework stores artifacts and all kinds of related documents in a database. Thus, queries may concern an artifact, but also the associated WHAT, HOW and WHY documents. We now give examples of queries on (a) sets of documented artifacts and (b) the documentation itself. Queries on non-documented artifacts can also be posed, but are not considered relevant to this paper.

4.1.1 Queries involving documented artifacts

According to our goal of integrating three types of documentation within a collaborative framework, three kinds of queries can be identified as far as artifacts are concerned: WHAT, HOW and WHY queries.

While the first type of query (WHAT queries) concerns finding out about the composition of artifacts, the second one (HOW queries) deals with the description of the production process that led to a certain artifact. Finally, WHY queries are concerned with the rationale behind an artifact, as described in Section 3. In the following examples, the queries are characterized according to the type of documentation involved – WHAT, HOW and WHY.

We point out that in many situations, the distinction among types of documentation is fuzzy, and depends on the users' vision of the world. Thus, for some people, an algorithm is a WHAT - metadata document, whereas for others it is typically HOW. By the same token, specification requirements may be considered WHAT but also constitute a complex explanation which is a node in a why-graph. Metadata are particularly useful to process time-oriented (when) queries. The queries that follow exemplify common needs of different types of groups which participate in an artifact's life cycle.

1. WHAT queries:
   (a) How does this gadget work? (i.e., retrieve its manual)
   (b) What are all algorithms from computational geometry used in this particular 3D-visualization software?
2. HOW queries:
   (a) How was this product built?
   (b) How was this GUI assembled?
3. WHY queries:
   (a) What was the chain of arguments that led to a certain product?
   (b) Why was module M1 chosen to configure this software?
   (c) What is the comparative quality of products P1 and P2 given the explanation chain behind their development?

4.1.2 Queries on documents themselves

Whereas the previous queries concerned documentation for a specific set of artifacts, interesting queries can also be posed on the stored documents themselves. This allows correlating all types of documents, e.g., to analyze market trends. This is only possible because they are all managed within a single database. Examples are:

1. When did a given production procedure stop being used?
2. When was Theory t made invalid and through which arguments?
3. Which are all documents that use diagrams of Type d?
4. What are all documentations written in English?
5. Retrieve all explanation chains that use explanation $E1$
6. How often is Theory t applied in the explanation for the construction of an artifact that is assembled using workflow $w1$?

4.2 Documented artifact (DA)

Intuitively, we define an artifact as composed of parts, which may themselves be artifacts. An artifact is hence built as any complex object, using the set constructor. However, semantics have to be associated with the use of this constructor in order to document the process of creating such an object. A *documented artifact* is any artifact which is associated with some kind of document (one or many in the WHAT, HOW, WHY collection). Documented artifacts may be composed of other (documented) artifacts. A *stored artifact* is an artifact that has been stored in a database. It has an associated reference.

We are now ready to describe the elements of the database: artifacts, documents and documented artifacts. Those are described using a syntax that is meant to be intuitive (symbols [ ] and { } represent respectively the tuple and set constructors, and symbols `/*` and `*/` encompass comments).

Non-documented artifacts

```
simple-artifact = [AttributeList] /* basic parts of an artifact */
artifact = simple-artifact              /* atomic artifact */
         | {artifact}                   /* complex artifact */
```

Documentation

```
documented-artifact = [artifact, {document}]
document [ ]
   /* documents are defined more precisely through what-, how-,
      why- types of documentation, as described thereafter  */
```

What documentation. WHAT documentation is *static.* It is created by recursively traversing the objects (artifacts) that constitute an artifact until atomic artifacts are reached and by getting all documentation associated with it.

```
simple-what-document =[AttributeList]
what-document = simple-what-document
             | complex-what-document
         /* A complex-what-document is built by applying constructors
            on simple (atomic) document constructors: set, list, tuple
            Its value is computed by going recursively into its
            substructures */
```

How documentation.
We consider HOW documentation to be defined by means of workflows. Thus, a HOW document is *dynamic* (in the sense that it can be executable by a workflow management system). A workflow is in fact a directed graph where nodes are activities and edges indicate execution dependencies among activities. There are some proposals to store and manage workflows within DBMS (e.g., [**?**]). They basically rely on three types of stored entities: *Activity* (describes a task); *agent* (those may execute tasks); *roles* (the role in which agent executes task).

```
How-atom = [activity, agent, role] | {[activity, agent, role]}
          | REL (How-atom,How-atom)
REL = AND | OR | XOR
Agent = person | system
Activity = manual activity | automated activity
How-document = {input} || Dependency-link(How-node) || {output}
How-node = How-atom | How-document
Dependency-link(How-node) = How-node SUCCEEDS How-node
                | How-node BEFORE How-node
                | SOMETIMES (How-node)
                | ALWAYS (How-node)
input  = artifact | documented-artifact | document | file | event
output = artifact | documented-artifact | document | file

/* REL indicates relationships among workflow task and Dependency links
denote dependencies among tasks, where
tasks may be atomic (how-atom) or encapsulate
complex workflows (how-document). The || symbol denotes concatenation.
*/
```

Why-documentation. The why-documentation is a set of why-graphs. It associates rationale with an artifact (atomic or complex), which leads to a why-documented artifact.

Figure **??** gives an example of a why-graph and the artifacts documented by this graph. As we can see from the figure, artifact A is composed of simple artifacts {a1, ..., an}. Participating artifact a1 can be explained either by E1 or by E2. In turn, E2 is explained by E3, which may be justified either by a complex explanation containing both E4 and E5 or by E6 . The symbol || denotes drains of the graphs ("terminal nodes").
We point out that each of these artifacts may have been conceived and developed by a different team in a collaborative environment. However, this becomes transparent in our framework, as all artifacts and their parts and documentation are integrated within the database.
Definition 4.1. A *whow-documented artifact* is a 4-tuple (a, WH, HO, WG) where a is a whow-documented artifact, and WH, HO and WG are sets of respectively WHAT, HOW and WHY documents.

4.3 Versioned artifact

Versioning in databases arises from the necessity of representing the real world. Versions are a means of storing different states of a given entity, thereby allowing the control of alternatives and of temporal data evolution. Versions are very common to any kind of cooperative environment, e.g., in the context of CASE systems and CAD/CAM projects, often for object-oriented databases (e.g., [**?**, **?**, **?**]). Versions are also often considered in the context of concurrency control (e.g., [**?**]) or as an explicit means to support cooperative work (e.g., [**?**]). Two versions are called *alternative* (or variant) if they are generated from the same original object. The support to alternative (or

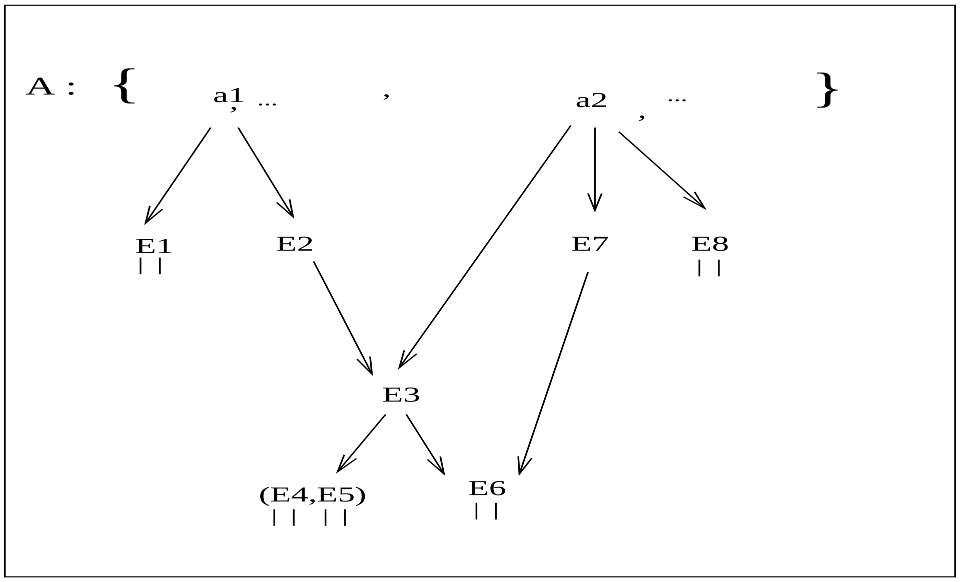


Figure 1: A why-documented artifact

parallel) versions allows, among others, testing different ideas, contrasting scenarios or meeting distinct design or user requirements.

Like any real world entity, artifacts are versioned. Intuitively, versioned artifacts are artifacts which have several versions. These versions are all connected to each other by some sort of link (physical or logical), which allows retrieving sets of versions of an artifact that obey a certain constraint (e.g., "Which are the compiler versions that were installed after a certain date?"). Since artifacts may be complex, their parts may also be versioned, and again "linked" to each other. This means that in the end the version database is organized in a very complex manner, where versions of artifacts and their components are all interlinked.

Figure **??** shows examples of complex artifact versions. An arrow represents a temporal evolution. On the top of the figure, x_i represents a version of x. x_2 is derived from x_1 (temporal evolution). x_5, x_3 and x_4 are alternatives (parallel evolution). The figure in the middle depicts an artifact x as composed of artifacts b and c. The graph at the bottom shows different versions for artifacts as well as for artifact parts. b_1 is part of both x_1 and x_4.

In order to solve this issue, we have chosen a particularly flexible versioning mechanism – the *database version mechanism* of [**?**], hereafter called *DBV* for short. It handles all complex links among versions as *logical links* in a way which is transparent to the user. This allows managing versions of artifacts (*and documents*), and connecting them all together into complex WHOW documented artifacts.

A conventional monoversion database (i.e., a database where versions are not considered) represents *one* state of a modeled part of the world. In the database version approach, a *multiversion database* simultaneously represents *several* states of the modeled part of the world. Whenever users want to handle one given version of an object, they actually just want to manage one of the states of the multiversion database. Intuitively, this is similar to the concept of *database view*, in which only relevant information is offered to the user. This state of the world within a multiversion database is therefore a logical view of the entire database, and is called a *database version*. Each database version in a multiversion database has an identifier v (i.e., each view can be uniquely

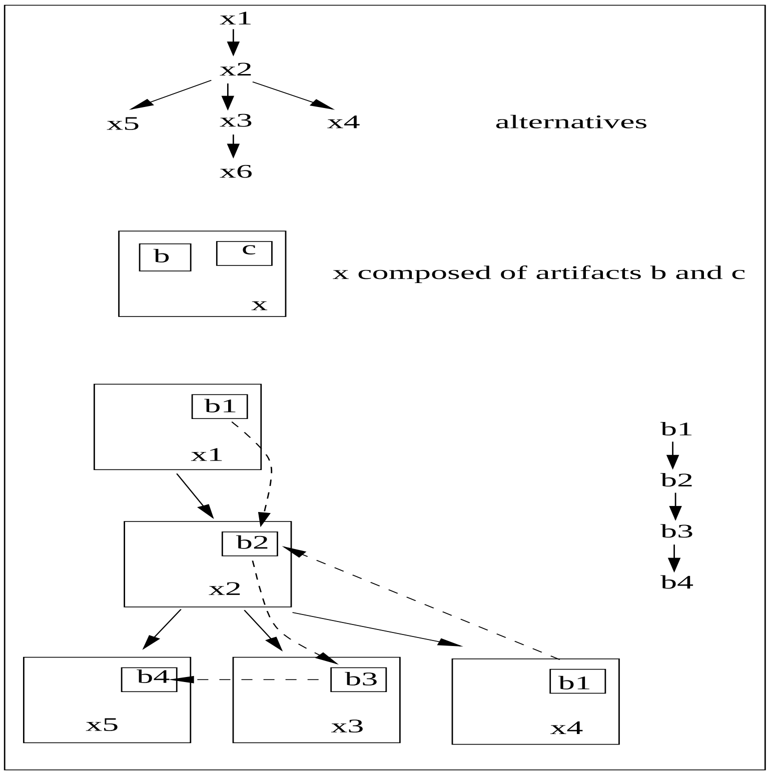


Figure 2: Arborescent artifact versions

identified). An object with several versions is called a multiversion object.

For the purpose of this paper, all we need to know is that the DBV mechanism relies on separating the logical database (a consistent state of the world) from the physical database issues, which are transparent to the user. Furthermore, the number of different actors collaborating in creating and managing an artifact version is irrelevant to this versioning discussion, since each group that participates in the development of a version can immediately have access to it, ignoring all others. Moreover, the DBV keeps track of derivation operations, thereby allowing the user to identify evolution trends, alternatives and other links across versions of an object. Updates of complex objects are handled in the same way as those of atomic objects. For more details on this mechanism and underlying model, the reader is referred to [**?**, **?**].

We recall that a *stored artifact* is stored in a database and has in particular an associated reference (e.g., object identifier in an object-oriented database). The DBV mechanism relies on identification of objects within each database version. Any multiversion object o_j has each of its versions identified by $(v_i,\ o_j)$, denoting the state of multiversion object o_j as seen in the logical database version v_j.

Definition 4.2. A *versioned artifact* is identified by a tuple (v, a), where a is a multiversion artifact and v is the identifier of a database version.

Definition 4.3. A *versioned document* is identified by a tuple (v, d), where v is the identifier of a database version and d is the identifier of a document.

Definition 4.4. A *versioned whow-documented artifact* is identified by the tuple (v, wda), where v is a database version and wda a whow-documented artifact.

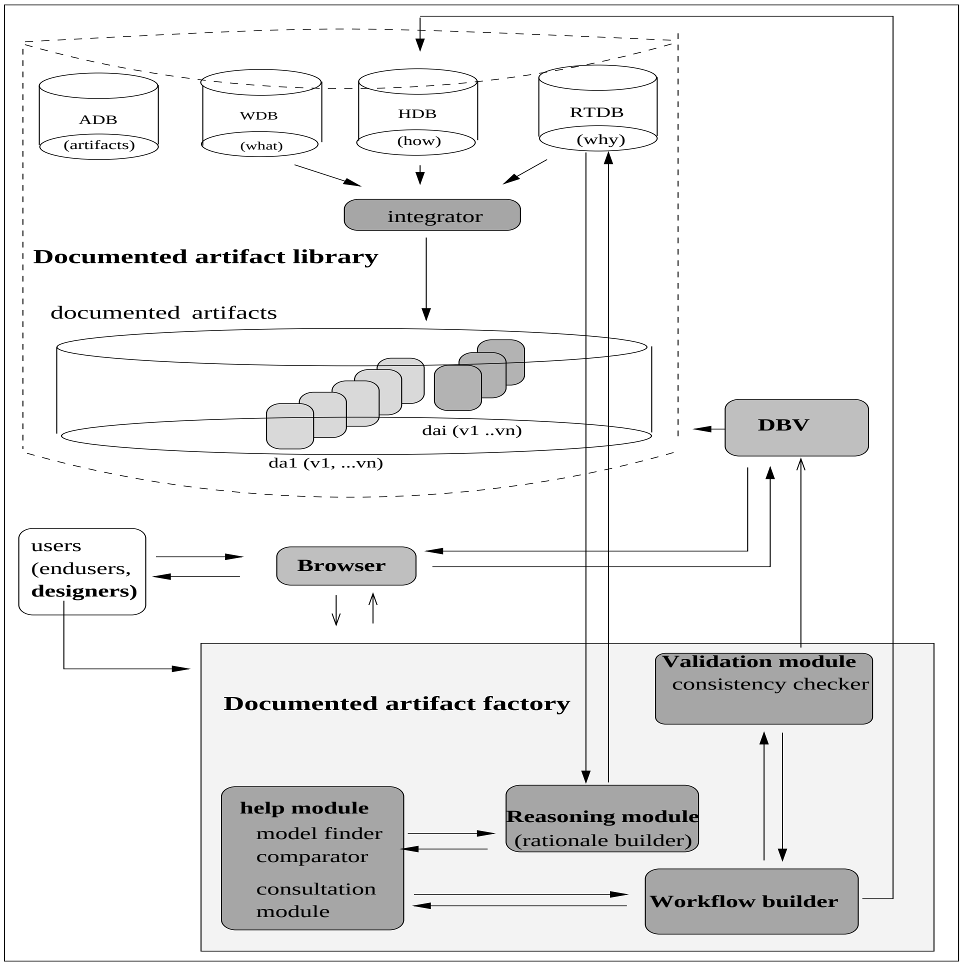


Figure 3: Architecture of WHOW

4.4 Architecture of WHOW

Figure **??** shows the architecture of WHOW. It consists of two levels: A database level, and a documented artifact construction and querying level.

The database level is managed by a DBMS which has the DBV mechanism. This DBMS manages four databases: An artifact database (ADB) containing stored artifacts, and three *documentation bases* – WDB, HDB and RTDB, containing respectively WHAT, HOW and WHY (rationale) documents. Objects of these four databases are integrated by means of an *integrator* module.

The construction and querying level allows the user to access the database and manage (versioned) artifacts, documents and documented artifacts. We refer to this large database of documented artifacts as **Documented artifact library**. The user may interact with this database in a querying mode through the *browser*, to pose queries such as the ones presented before.

Alternatively, the user can construct new documents, and attach documents to artifacts (creating WHOW documented artifacts) by combining objects stored in the four databases, or inserting new objects. This is achieved via the **Documented artifact factory**, a toolbox for interacting with the DBMS. In particular, the factory has tools for construction of why-graphs (*Rationale builder*) and workflows (*Workflow builder*). The interaction with the factory is assisted by the *help* module and validated by the *validation* module.

Description of our work in implementing some of these modules, and on the DBV mechanism, is outside the scope of this paper and can be found elsewhere [?, ?, ?].

5 Example of using the framework

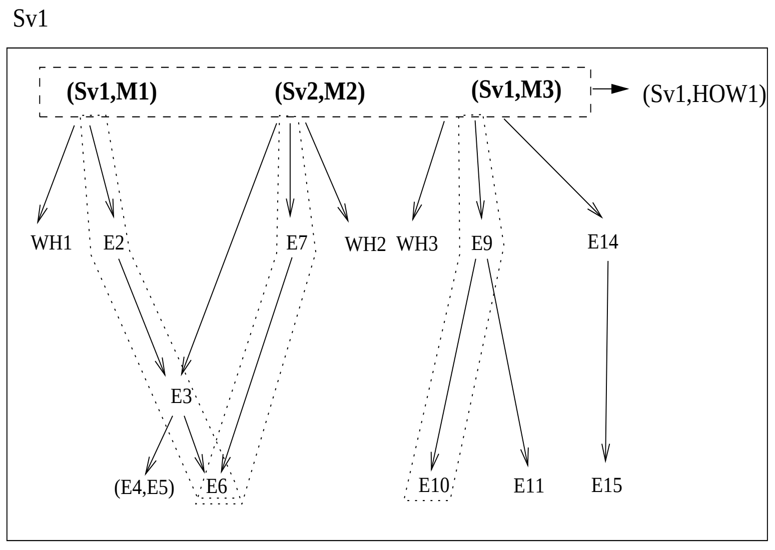
We now give an example of how an artifact is stored within WHOW. Let us consider a complex documented artifact SW – a piece of software, which is being designed and built by several teams. SW is assembled by putting together three modules (also documented artifacts) $M1, M2, M3$ – a text editor, a compiler and a graphical interface. Figure **??** shows two logical database versions of SW, Sv1 and Sv2, indicating the different documentation links among its modules, and how they would be maintained within our framework. SW is assembled by integrating the text editor, compiler and interface through a set of steps documented in a HOW workflow document HOW1. In this figure, database elements are identified by pairs (version identifier, object identifier) – e.g., (Sv1, M1) is the version of the text editor used in the Sv1 version of SW. In order not to burden the figure, we did not indicate the full identifiers in all cases, but the reader can assume that all node labels are pairs (Svi, Dj) where Svi is the identifier of the database version and Dj indicates the document – e.g., (Sv1, HOW1) in the figure.

Thanks to the DBV mechanism, each software version can be managed by different users or user teams independently without considering its relationships with the other versions. Thus, for all effects, a given team may work with Sv1 or with Sv2, or even with both alternately, but does not need to worry about versioning. Module M1 is documented by a WHAT document WH1 (a manual) and one why-graph where just just one explanation path (E2, E3, E6) is chosen both in Sv1 and Sv2. Module M2 has a WHAT document WH2 (compiler installation instructions) – again we recall this may be versioned or not! M2 has again one single why-graph, but distinct explanation chains have been chosen for it according to where it is managed, Sv1 or Sv2, respectively (E7, E6) and (E3, E4, E5). Finally, M3 has one WHAT document WH3 (interface specification), but now it has one explanation chain in Sv1 (E9, E10) and *two* explanation chains in Sv2 (E9, E11), and (E14, E15).

We point out a few interesting issues which occur in real life and can be supported in our framework. First, different artifacts and artifact versions may share documentation. Second, why-graphs may share explanations (a novel aspect in rationale management) since explanations are database objects (e.g., modules M1 and M2 share explanations E3 and E6 in their why-graphs). Third, we can easily manage configurations of versioned artifacts and the associated documentations, thanks to the orthogonality between the DBV version manager and the entities being versioned (since this manager treats all kinds of database objects as versionable entities). Fourth, the existence of teams cooperating to create the artifact is managed by the framework by integrating documentation facilities with the versioning mechanism, presenting a unified view to all actors involved. At the same time, different teams may work with distinct parts or versions of the artifact, thanks to the concurrency control functions provided by the underlying database management system.

Finally, one single artifact version (M3) can have more than one explanation chain associated.

This last comment merits a longer discussion, since it is counter-intuitive to allow alternative WHY documents to one single artifact. In real life, this situation is found in at least two cases. The first case occurs when the explanation is constructed by teams



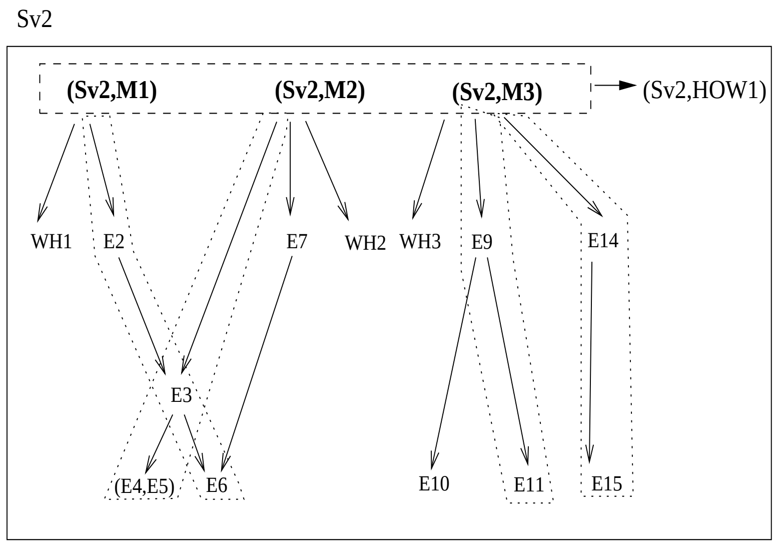


Figure 4: Versioning complex whow-artifacts

that did not participate in the construction of the artifact (e.g., reverse engineering). Thus, they may have distinct acceptable justifications of why a given artifact is/behaves in a certain way (see [?] for an example in interface design rationale). The second case occurs when only certain aspects of an artifact are emphasized in order to explain its creation – e.g., functionality issues versus performance issues – and correspond to multiple artifact interpretations.

6 Conclusions

This paper presented a database-centered framework for managing documentation generated in cooperative work, that unifies all kinds of artifact documentation, thereby allowing smooth transition between different stages of an artifact's life cycle, and more efficient maintenance, and communication among ¡ participants of cooperative work. The main idea is to manage all types of documentation (WHAT, HOW and WHY) within a single database, supported by a database version mechanism. The architecture separates the artifact itself from the issues of reasoning and discussion to arrive at an artifact (WHY), the procedures to produce the artifact (HOW), and the artifact description (WHAT). This permits independent evolution of documentation base and artifact base.

Data (the documents) are managed by the DBV version mechanism, which ensures proper handling of configurations and alternatives, which are managed orthogonally to the database stored data. Reasoning is supported by why-graphs, and workflows document HOW to assemble an artifact. Documents may evolve independently from the artifacts themselves. This allows the versioning mechanism to keep track of the validity of theories used to construct a product. The framework presented here is modular in the sense that it allows all types of combinations of documentation associations. While some applications require intensive use of WHAT- and WHY-documentation simultaneously (e.g., geologic map creation), other are just concerned with WHAT- and HOW-documentation.

Several extensions are being considered, both in terms of implementation and in terms of theoretical issues. A preliminary experience in why-graph implementation is the work of [?] on geological hypermaps. The use of workflows as HOW documents managed within DBMS was tested within a system constructed for the domain of environmental applications [?]. We intend to couple these prototypes to the DBV implementation and associate other kinds of documents.

One strong assumption in WHOW is that stored why-graphs have been constructed by the people who participated in creating and designing the artifacts. However, it is an open issue of how rationale can be efficiently captured. Indeed, documentation of a rationale is very labor-intensive. It is our belief that our proposal for a rationale database RTDB will help users organize their WHY documentation, thus allowing the incremental construction of a rationale knowledge base. Other open issues along the same line include a discussion on various types of rationale (e.g., not only objective criteria, but also social or cultural issues may have to be taken into account) and the fact that rationale is domain-specific.

Acknowledgements

This work was developed with Brazilian grants from CNPq, FAPESP and the Pronex/Finep program SAI (Advanced Information Systems). The authors thank Jacques Wainer for pointing out the connection of this research with that on design rationale as well as the Berlin-Brandenburg Graduate School on Distributed Information Systems for its travel support.

References

[1] G. Alonso and C. Hagen. Geo-Opera: Workflow Concepts for Spatial Processes. In *Advances in Spatial Databases (Proc. of SSD'97)*, pages 238–257, LNCS No. 1262, M. Scholl and A. Voisard (Eds.), Springer-Verlag, Berlin/Heidelberg/New York, 1997.

[2] A. Ailamaki, Y. Ioannidis, and M. Livny. Scientific Workflow Management by Database Management. In *Proc. 10th IEEE International Conference on Scientific and Statistical Database Management*, pages 190–201, 1998.

[3] J. Conklin and M. Begeman. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331, 1988.

[4] W. Cellary and G. Jomier. Consistency of Versions in Object-Oriented Databases. In *Proc. 16th VLDB*, pages 432–441, 1990.

[5] A. Dattolo and V. Loia. Collaborative Version Control in a Agent-based Hypertext Environment . *Information Systems*, 21(2):127–145, 1996.

[6] S. Gancarski and G. Jomier. Managing Entity Versions within their Contexts: a Formal Approach. In *5th International Conference,Database and Expert Systems Applications DEXA94*, pages 400–409, 1994.

[7] O. Günther and A. Voisard. Metadata in Geographic and Environmental Data Management. In W. Klas and A. Shet, editors, *Managing Multimedia Data: Using Metadata to Integrate and Apply Digital Data*. Mc-Graw Hill, 1998.

[8] R. H. Katz. Toward a Unified Framework for Version Modelling in Engineering Databases. *ACM Computing Surveys*, 22(4):375–408, 1990.

[9] F. Llirbat, E. Simon, and D. Tombroff. Using Versions in Update Transactions: applications in Integrity Checking. In *Proceedings VLDB 1998*, pages 96–105, 1998.

[10] M. Manouvrier. Versions d'Objets de Grande Taille: Répresentation, Comparaison et Mises-à-Jour. PhD Thesis Internal Report, University Paris-Dauphine, Feb 1999.

[11] W. Michener, J. Brunt, J. Helly, T.Kirchner, and S. Stafford. Nongeospatial Metadata for the Ecological Sciences. *Ecological Applications*, 7(1):330–342, 1997.

[12] T. P. Moran and J. M. Carroll, editors. *Design Rationale: Concepts, Techniques and Use.* Laurence Erlbaum Associates, 1996.

[13] T. P. Moran and J. M. Carroll. *Overview of Design Rationale*, Chapter 1, pages 1–20. In [**?**], 1996.

[14] C. B. Medeiros, G. Vossen, and M. Weske. GEO-WASA – Combining GIS Technology and Workflow Management. In *Proc. of the 7th Israeli Conference on Computer-Based Systems and Software Engineering*, pages 122–139, 1996.

[15] J. Meidanis, G. Vossen, and M. Weske. Using Workflow Management in DNA Sequencing. In *Proc. First IFCIS Conference on Cooperative Information Systems*, 1996.

[16] A. MacLean, R. Young, V. Bellotti, and T. Moran. *Questions, Options and Criteria: Elements of Design Space Analysis*, chapter 3, pages 53–107. In Moran and Carroll [**?**], 1996.

[17] M. Rusinkiewicz and A. Sheth. Specification and Execution of Transactional Workflows. In W. Kim, editor, *Modern Database Systems. The Object Model, Interoperability and Beyond*, pages 592–620. ACM Press, 1995.

[18] L. Seffino, WOODSS - Workflow based Spatial Decision Support System, Masters' thesis, UNICAMP, 1998 (In Portuguese)

[19] G. Talens and C. Oussalah. Version d'objets pour l'ingénierie. *Technique et Science Informatiques*, 15(2):145–178, 1996.

[20] A. Voisard. Abduction and Deduction in Geologic Hypermaps. *Advances in Spatial Databases*, Proceedings of the 6th International Symposium on Spatial Databases (SSD'99), Lecture Notes in Computer Science No. 1651, R. H. Güting, D. Papadias and F. Lochovski (Eds.), Springer-Verlag, Berlin/Heidelberg/New York, 1999.

[21] W. Wieczerzycki and J. Rykowski. Version Support for CAD/CASE Databases. In *Proceedings East/West Database Workshop*, Workshops in Computing, pages 249–260. Springer Verlag, Berlin/Heidelberg/New York, 1994.

[22] J. Wainer, M. Weske, G. Vossen, and C. B. Medeiros. Scientific Workflow Systems. In *Proc. of the NSF Workshop on Workflow and Process Automation Information Systems*, 1996.