CWSLR Model Used to Synthesize a Software Development Environment and an Application Design Tool

Moisés González-García¹,²,
Ana Ma. Martínez-Enríquez¹,
Raul Jacinto-Montes²

¹Centro de Investigación y Estudios Avanzados • I. P. N. (Ciudad de México),
Av. I.P.N. No. 2508, Mexico D.F.
²Centro de Investigación y Estudios Avanzados • I. P. N. (Guadalajara),
Av. Lopez Mateos sur No. 590,45090 Guadalajara, Jalisco México
³Centro Nacional de Investigación y Desarrollo Tecnológico
Internado Palmira S/N, 62490 Cuernavaca, Morelos México

Abstract A Software Engineering environment, the Cooperative Work Software Process (CWS), and a design tool, the Cooperative Work Application Design (CWAD) are described. Both are based on the Cooperative Work Storage, Logic, and Resources model (CWSLR). A simplified version of the model is used to drive the user interface, and the structure of the model determines the control flow of the design process supported by CWAD. Utilization of the model is enabled because its components are explicitly related in abstraction hierarchies, and each dimension elaborates on simple predefined structures.

Keywords: CSCW, IS, distributed systems, open systems, distributed artificial intelligence

1. Introduction

The software development process (SDP) is a domain where cooperative work is the only way to succeed. This process obtains the artifacts that describe the sequence of models needed in the development of an information system. The work involved in SDP can be organized in several workflows [5] needed to build a software product.

To support the SDP several environments have been proposed [16, 17, 18, 20, and 21]. Also many software development tools [19] have been proposed. We describe in this paper the mechanisms used to synthesize from the CWSLR model [1, 2, and 3] to both a software-engineering environment named Cooperative Work Software Process (CWSP)(Section 3), and a design tool named Cooperative Work Application Design (CWAD)(Section 4). The fact that both software applications are based on a model obtained in a domain analysis of cooperative work information systems (CWIS) [3] suggest that they should support a cooperative work process, as we explain in sections 3, and 4.

Beginning the design process by the definition of a model is enforced by the ideas introduced by Bass, et. al. [4], where they recommend use a reference model to initiate design of a software application. A reference model as used by these authors is a standard decomposition of a known problem into parts that cooperatively solve the problem.

This paper primary focus is in CWAD design tool. But during the design process became obvious that the tool has to be part of a software engineering environment (SEE), and the whole SEE should be concordant with CWSLR model. That is why a Framework Incremental Life-Cycle approach [14] is used. In this approach, as first task, sufficient overall requirements specification and system design is performed to provide a framework for

---

¹This paper has been developed with a doctoral scholarship provided by CONACyT México
incrementation (i.e. CWSI). Increments are then produced iterating the workflows included in a cycle (i.e. CWAD).

The paper is organized as follow. Section 2 presents an abstract description of CWSLR model, giving the required knowledge to understand following sections. Section 3 describes the SEE providing a framework for incrementation. Section 4 shows CWAD tool details. Section 5 presents some computational requirements for both software products (CWSP, and CWAD) giving a current technology state, and section 6 presents conclusions.

2. Simplified CWSLR model.

CWSLR model lies three dimensions, each one lies in turn five abstraction levels (figures 3, and 4). This set of structures gives form, rationale, and rules to develop an information systems supporting cooperation (CWIS).

CWSLR model considers all components of most known models in the areas of open distributed systems, and object oriented management as shown by authors at PDPTA’99 [2].

The CWSLR model definition begins synthesizing its general architecture and the design of the structure of Storage Dimension [1]. A second stage carries structure homogenization of the dimensions and the validation of the model with other relevant models [2]. A third stage refines the three dimensions of the model to a similar detail where the main structure and concepts that enables them are mentioned, and related [3].

CWSLR model considers information systems as an application that can be constructed in turn by one or several applications (e.g. ODBMS, communication services, payroll, acquisitions, etc.). The functionality and data of the application should be stored in STORAGE-D dimension (one of the three dimensions assembling that model).

There is in CWSLR model the concept of a minimal application that gives a bottom-bound to the construction process. A minimal application includes a functional level (one of the six functional levels included in CWSLR model, indicated by arrows), and at least an adjacent input or output structural level (one of the nine structural levels included in CWSLR model, indicated by black circles). There are unary or binary minimal applications, as shown in figure 1.

![Minimal applications](image)

Figure 1. Minimal applications.

When the model has many components, developers can use abstraction to reduce the number of details to be considered. This is the case with CWSLR model, in figure 2 it is reduced to six sets of applications, two sets for each dimension. Those sets of applications, refined as a hierarchy of four levels, become an Applications Type Catalog (AC). AC is used to control CWSP’s interaction with users. A detailed structure of CWSLR model is used to drive the control flow of CWAD, as will be mentioned in section 4.


In this section we give the main characteristics of CWSP environment, that become context of the components that operate in it. The principal features are developed incrementally to build a test-bed for the development of CWAD tool.
In the process to define CWSP, are included several characteristics of CWIs [3] into the software development process environment. The resulting SEE should support the following requirements, giving sufficient understanding of the cooperative work behavior, organized into project management, process management, and storage management, as follows:

**Project Management.**
1. Work is managed using projects, which can be associated with one or more main-applications.
2. Project work is managed using a work breakdown structure [8] composed of units. The results are organized as shared Unit Development Folders (UDFs) [9]. These folders are reviewed at predefined milestones [5].
3. There are personal notebook, and UDF configuration management processes.
4. When the project terminate, there should be the software products (systems or main-applications) and a final report.

**Process Management.**
5. Process is organized in pliases and workflows, as the unified software development process USDP, proposed by Jacobson, et. al. [5].
6. It is proposed, that any artifact obtained by the process should be developed by groups in the following modes:
   a) Each member of the group working individually,
   b) The group working collaboratively, focussing on group awareness (procedure COL1).
   c) The group cooperating, with focus on conflict resolution (procedure COP1).
7. Every activity in a workflow should use the same eight problem-solving steps e. g. problem definition, value system design, synthesis, analysis, prototyping, optimization, evaluation with value system, and planing, as proposed by Gonzalez-Garcia [6].
8. Refining the problem solving steps, a set of unit operations is obtained e.g. partition, aggregation, replication, etc. [4, 7]. This set can be used to propose a set of tools [11], that can be used in any workflow of the development process.

**Storage Management.**
9. The artifacts produced, by the resulting breakdown, are organized as a set of (UDFs) with content based on work done by Ingrassia [9].
10. Each individual working in a project has a personal folder or multimedia notebook [10]. Notebook pages can be exchanged with colleagues.
11. As the work continues, information in personal folders, is gathered, sifted, sorted, reorganized for presentation and inserted into shared UDFs.
12. Each submission of UDFs contains revised and reorganized information from previous versions.
13. The final report contains not only all the UDFs, but also captures the workers experience, decision making process [10, 12, 13], and knowledge relating to the project (including ideas pursued and ultimately abandoned).
CWSP is a SEE guided by an application catalog (AC) that can be the initial structure for a reuse library. The mentioned AC was drawn from the CWSLR model, considering that it is the result of a domain analysis for CWlSS. From this point of view the three principal elements described in this paper: 1) CWSLR model, 2) CWSP SIX, and 3) CWAD tool, are aimed to the trends mentioned by Schäfer et al. [15] to reach closure of the reuse problem:

1. Domain analysis and domain engineering provide the means and support to implement preplanned reuse;
2. Software reuse processes integrate reuse into software development; and
3. Mega-programming which is proposed to establish a paradigm of software development: domain specific, reuse-based, and process-driven.

This reuse perspective of the AC metaphor has become obvious. This fact and the coincidence between the proposed SEE requirements, and Schäfer's trends enforces that the results obtained are good and can be usefull.

AC has initially a four level hierarchical structure, organized in three areas, one for each dimension of the CWSLR model. It has six elements in the first level, twenty-three elements in the second level, eight elements in the third level, and sixty-four elements in the fourth level. This initial structure enables the use of AC to guide the process. But AC manipulation should allow updates, and be very flexible, it should be possible to add or delete application types, in any level of the hierarchy, and also add or delete levels to the hierarchy.

CWSP and every workflow tool (as CWAD), uses AC to establish the type of application to work on.


CWAD is an architecture design tool that supports a process to obtain the software architecture of a computing system. The software architecture as defined by Bass et al. [4] is the structure or structures of the system, which comprises software components, the externally visible properties of those components, and the relationships among them.

CWAD supports the work of the architect following the structure of CWSLR model [1, 2, AND 3], and including all the relations and restrictions that this model imposes. CWAD simplify handling the complexity of the CWIS architectural design visiting, in a bottom-up approach, each of the predefined applications that can be included in STORAGE-D dimension. The application types are determined using the AC. This catalog is used as a menu, to makes possible several different sequences to process the applications. One of the alternatives is to proceed in the order specified by the architect.

CWAD shows four major components and a set of auxiliary applications (figure 5). Its procedure reads the application types from AC, giving options to the architect, and guiding the architecture design. The numbers 1, 2, and 3, shows the order of the process for each application.

CWAD process granularity unit is an application, as mentioned before. An application can have applications as components, and those applications can be related in any possible way. When the templates (questionnaires), about STORAGE-D are finished, the control is transferred to the RESOURCE-D manager. From RESOURCE-D manager, when the corresponding templates are finished, control is transferred to LOGIC-D manager to collect the answers required in templates for this dimension.

Component "Management of STORAGE-D" (figure 5), cover only the structural vertical of STORAGE-D (figure 3, a). Component "Management of RESOURCE-D" cover the horizontal structures relating dimensions STORAGE-D and RESOURCE-D, at their three structural levels (figure 4) and the vertical structure of RESOURCE-D (figure 3, b). Component "Management of LOGIC-D" dimension covers the horizontal structures.
relating the three dimensions, at the three structural levels (figure 4) and the vertical structure in LOGIC-D dimension (figure 3, c).

When the three sets of templates are finished for the application in turn, the control is returned to the component “Management of Design Process”, which continues, the process with the next predefined application type. This procedure is finished when the architect decides that the applications are complete. The applications are complete when the desired predefined applications for the three dimensions and any new applications have been processed.

Figure 3. CWSLR, structure of dimensions.

Figure 4. CWSLR, horizontal structure.

Some of the templates, can allow to introduce a link to artifacts (example: diagrams, tables, and prototypes). These links to artifacts can be: 1) a file containing the artifact, with the extension of an auxiliary application, 2) open any auxiliary application to create the artifact, 3) the location of the artifact in non digitized form, 4) leave, the link to artifact, empty. Option 4 means that in the future the architect can chose either option 1, 2 or 3 to complete the information about the artifact.

In CWAD process, it is valid to have empty predefined applications (non-existing in the computing system) and to have new types of applications (different to those predefined in the AC) in any functional level.

When all the predefined types of application of the catalog have been processed, there is an option to add new types of applications. This option is also in the first display of the process for a predefined application, making possible the description of a new type of application instead. When the process for the new application is finished, the CWAD design process manager will try to process the predefined type of application that was not processed when the architect decided to process a new type of application.

The design process is incremental, so an application can have releases. The next release of an application can have more applications, or a different set of them.

When CWAD’s “Management of design process” module finishes the process of the whole set of predefined applications, and new ones, then control will be transferred to the RESOURCE-D manager. This manager steers the description of the structure and type of resources of the network where the computing will execute.
RESOURCE-D finishes, control is transferred to LOGIC-D manager to complete description and documentation of the entire computing system (the computing system seen as an application made of applications).

![Control flow diagram for CWAD](image)

Figure 5. Control flow diagram for CWAD.

When CWAD process finishes there is an option to produce a table that shows in column one all applications described and documented, and in column two all applications predefined in CWSLR. This table will make explicit the existence of empty and new applications. In the same output, there is a list of empty applications with the flag "incomplete CWIS", and a list of new applications with the flag "Deviations from CWSLR".

CWAD allows three different processes for an application:

1. An in-house developed application designed and documented using CWAD process.
2. An in-house of the shelf application (previously designed or described using CWAD). It can be an application or a different stand-alone application.
3. A commercial application, where the three dimensions essential questionnaires can be answered, and the existing on-line documentation (about software architecture) can be stored. The flexibility is critical, allowing partial information.

5. Computational Requirements.

It is proposed, and been evaluated that CWSP and CWAD use a desktop environment based on Java [22], and conforming to the Common Desktop Environment (CDE) [23], recognized as a standard for open systems. CDE is based on Motif including standard applications, and a front panel for quick access and launching of applications and support for multiple workspaces.

CDE support for multiple workspaces enable to change naturally among modes:

a) Each member of the group working individually,

b) Group working collaboratively, and
c) Group cooperating.
Development can use a Java-based Web server [24], and an object request broker [25]. Enhancing portability, extensibility, efficiency, security, and ease of creation.

6. Conclusions.

The model CWSLIZ was simplified to use it as the structure for the user interface, and a driver for simple sequence of tasks.

During the design of CWAD tool it became obvious that it was needed an environment to give context to the tool as support for one of the five workflow defined in the USDP. CWSI' software engineering environment is proposed, and its cooperative work characteristics are presented.

The development process for CWSI and CWAD, has to be incremental non-monolithic, following a Framework Incremental Life-Cycle.

The sequence of operations identified for both applications are simplified using the two abstraction hierarchies of CWSLR; a) vertical bottom-up, and b) horizontal left-to-right. Both hierarchies increment in abstraction following the arrow heads (figures 3, and 4).

The process of STORAGE-D is centered in outputs (results), and user options. The process of RESOURCE-D is centered in networks of resources: human, and material. The process of LOGICAL-D is centered in software architectures, business organization, and cooperation. These well established focuses for each dimension makes feasible use of templates (questionnaires), at the user interface.

Working on technology based on Java carries all its natural advantages.

References


[9] Ingrassia Frank S.: The Unit Development Folder (UDF): A Tell-Year Perspective; EI0263-4/87/0000/04055@0.00, 1987 IEEE.


Hawaii International Conference of System Sciences (HICSS 94), pp. 191-200.


[17] Ramamoorthy C. V., Usuda Yutaka, Tsai Wei-Tck, Prakash Atul; GENESIS : An Integrated Environment for Supporting Development and Evolution of Software; 0730-3157/85/0000/0472$01.00, 1985 IEEE.


