CAP

The CyberSpace Architecture Project

Introduction

How to Survive in the CyberSpace

*This paper shows the basic concepts and components of the CyberSpace Architecture Project (*CAP*). The first part introduces the main ideas, while the second part describes the CyberSpace Foundation (*CSF*), the CyberSpace Object Architecture (*COA*), Object Oriented Views (*OOV*) and gives some notes on how to use* CAP *best.*

1 Introduction

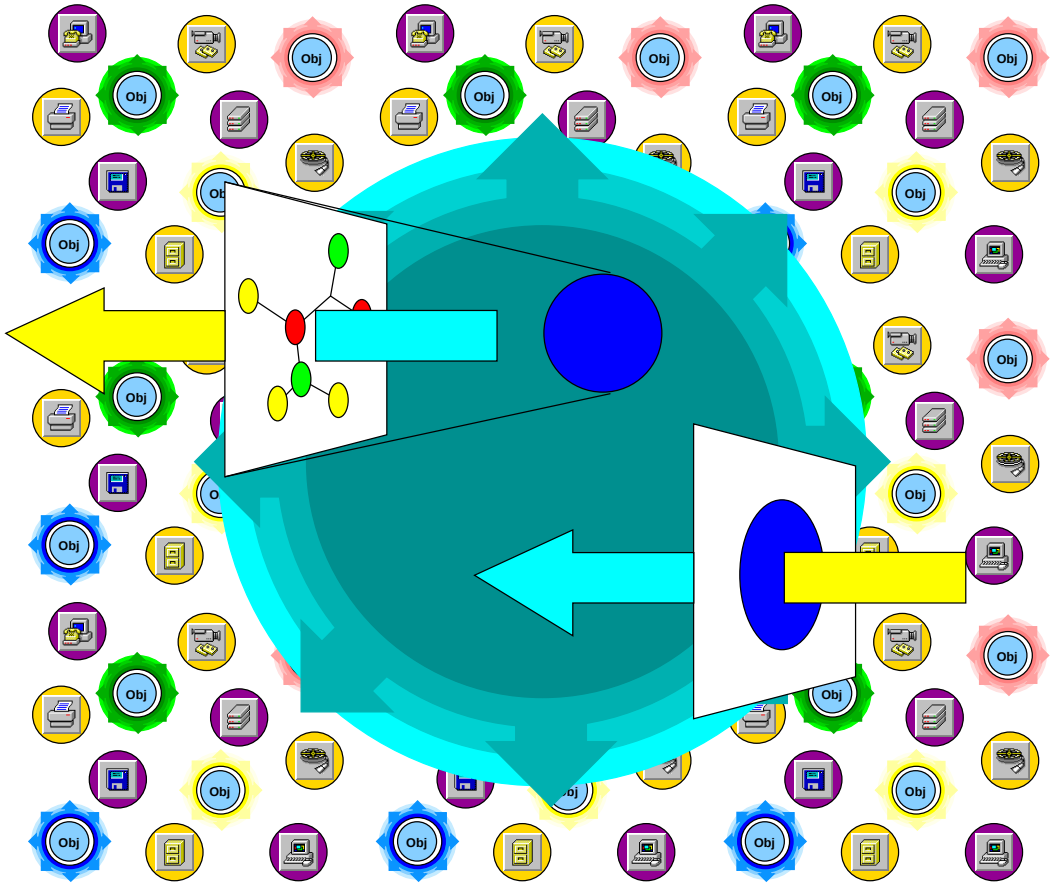
Today's open and complex environments, characterized by catchwords like heterogenous, distributed platforms, multiple languages, worldwide networking connections, changing software environments, unforeseeable new locations and future conditions, multi media and evolving systems form a highly complex, enormous big and evolving wolrd wide distributed data processing system: the *CyberSpace*[1].

Currently there is a big gap between these various evolving sophisticated modern technologies on the one hand side and users with their real world problems on the other, not knowing how to build on that kind of quicksand.

The goal of the CyberSpace Architecture Project (CAP) is to provide flexible means to screen respective areas from each other, nevertheless enabling flexible access from side to side.

To achieve this goal, a consequent application of object oriented design concepts is used within CAP, to provide flexible mechanisms for encapsulation of inner components, excapsulation of the outer world and the provision of highly abstract, general and commonly useful building blocks; see the following figure.

[1]The term CyberSpace was originally introduced by William Gibson in his science fiction "Neuromancer". It is sometimes used only for high quality visualisations, which is not meant here.



A software component (object) embedded in an environment made of several further components, floating around in CyberSpace. Access from the inside to the outer world is made through an interface (top left), a "view" of the outer world. Correspondingly, the object itself is accessed from the outside world through an interface (bottom right), representing a "view" of that object.

These mechanisms allow to create clearly defined software components through the provision of frames focussing on the relevant aspects - increasing the number of reusable system components - and to access such components from the outside through intelligent, flexible shells.

The CyberSpace Architecture does not provide yet another closed layer requiring to build every software on to take benefit. Instead, flexible and movable building blocks are provided, which are prepared to coexist with other components, prepared to be changed or replaced.

All in all the CyberSpace Architecture tries to integrate various modern instruments and technologies into a flexible frame made of intelligent, abstract interfaces, providing easy and expandable access to the various resources available.

2 Main Ideas

This section presents the main ideas which guided the design of CAP.

2.1 The Part and The Whole

Openness means being accessible if and where necessary.

Completeness in conjunction with openness requires to build *integrated* systems, whose components are separable.

Cooperativity of components requires them to be easy *integratable*, while cooperativity of integrated systems means to make it easy to separate cooperative components.

Good systems and their components are open, integrated, inegratable and cooperative. Their design reflects the fact, that nothing is unchangeable, but the task they shall perform is done well.

To build such systems, it follows the necessity to carefully create the system components, to ask which parts are component- and which are systemspecific and to apply object oriented design guidelines to structure the system on all imaginable levels.

2.2 Subject Orientation

Shifting the focus from technologies to whole systems and their environments emphasizes the role of tasks and users.

Subject Orientation means to concentrate not only on the objects to build, but also on the users of these objects, their views of them and their needs.

Subject Orientation is for one thing a direct consequence of applying object oriented design principles and for another an important addition to object orientation.

2.3 Capture, Don't Code

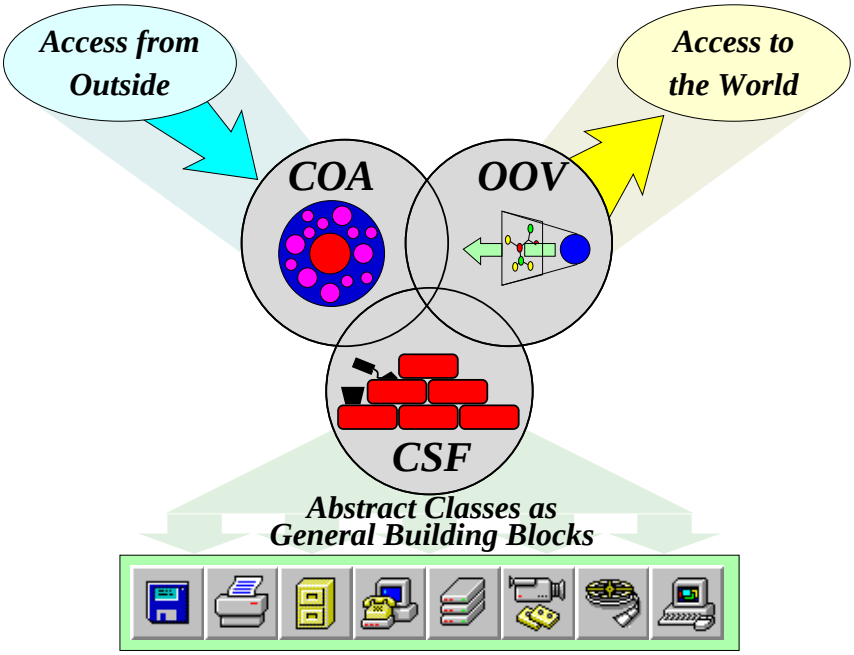
"Capture, Don't Code" is an old design principle: Try to capture the things as they are given instead of coding the consequences, i.e mixing what you want to describe with the properties of a specific language. This mixture renders it more difficult to reuse the coded information or to do changes.

A problem oriented description is desirable, allowing to describe facts without regard to concrete solutions or implementations. Ideally, such a description can be used to generate a solution automatically.

Naturally, no language is really "general". But there are more appropriate and less appropriate ones. Thus, providing – probably several – languages and concepts for formulating problems as descriptive as possible helps to increase maintainability and reusability.

3 The Architecture

The components of CAP*: the CyberSpace Foundation (*CSF*) provides basic building blocks, Object Oriented Views (*OOV*) serve as interfaces to the outside world while the CyberSpace Object Architecture (*COA*) provides flexible interfaces to others.*



The CyberSpace Architecture is based mainly on the components shown in the above figure. Specifically, these are[2]:

CSF - **The CyberSpace Foundation**
: captures various aspects of software and resources and provides access to them in a primarily very general form. It abstracts from various usually uninteresting and implementation or system specific details and therefore allows to focus on the relevant aspects. There are no technical elements contained, but basic components to a high degree on a purely logical level.

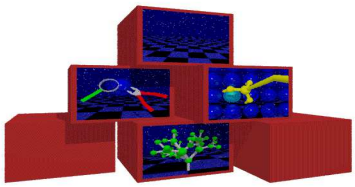
COA - **The CyberSpace Object Architecture**
: describes a model for objects designed to meet the requirements given by todays open and complex systems and environments. It is based on and extends the current object model, thereby introducing degrees of freedom between design and programming and fits seemlessly into it. COA allows to access objects in a spohisticated, flexible manner.

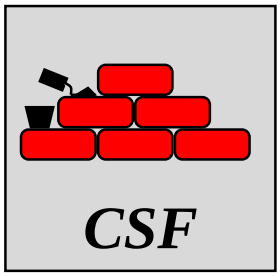
OOV - **Object Oriented Views**
: support the integration of different views of users into object oriented systems. Such views are a direct consequence of applying the principle of subject orientation to object oriented modelling. They serve as filters for perceiving the world, providing a platform to build specific applications on.

These components are described in the following sections.

[2]For details refer to the respective documentation ([?], [?], [?])

3.1 The CyberSpace Foundation





The CyberSpace Foundation is the base of CAP. It provides a set of fundamental terms for the design of software systems in form of highly abstract base classes.

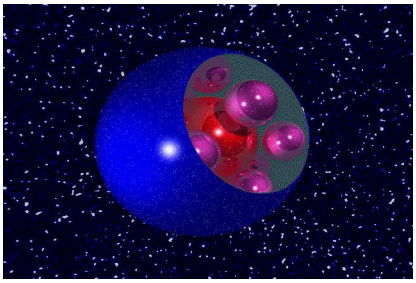
The CSF captures a wide range of aspects of software systems such as data structure (bags, sets, lists etc.), presentation and manipulation (user interfaces), persistency (data bases), spatial aspects, location etc.

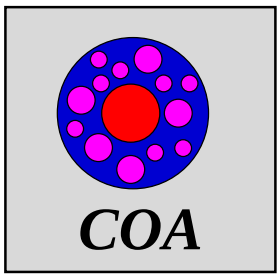
The design of the CSF does not focus and is not bound to any specific implementation. Instead the basic properties of the "CyberSpace" itself are examined. This leads to a high degree of generality and therefore portability of the CSF classes.

This basic CSF interface allows to encapsulate several specific services like more technical standard class libraries, user interfaces, data bases, communication libraries etc.

The CSF serves as an anchor for these services. The access to more specific features is of course possible by using derivation and other object oriented features.

3.2 The CyberSpace Object Architecture





The CyberSpace Object Architecture (COA) is a straightforward extension and improvement of the common object model. It is based on and made of ordinary objects, and COA objects just look like these, making integration of COA objects easy. COA is an object architecture, providing services and a framework for building sophisticated objects.

COA objects are designed to meet the requirements given by todays complex and open systems. They are flexible, adaptive, they have dynamic interfaces and are prepared for porting, even at runtime.

The concept of COA is to focus from the start on openness. Objects placed in class

hierarchies and concrete systems should not be bound there, as the hierarchies and the systems may change while the objects shall persist.

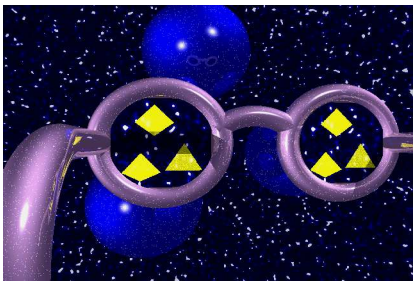
If system requirements cause too much tension on an object, the results of modeling it will most likely be unsatisfying, leading to either overloaded or uncomplete models.

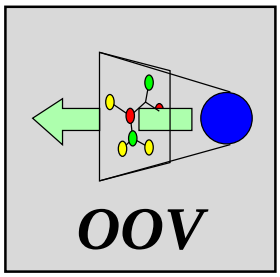
COA provides means to relax the tension caused by numerous system requirements, without disregarding parts of them.

The equipment with rich information in a highly generic fashion enables COA objects to react flexible and adaptable to their environment.

Using COA can save work, resolv design conflicts and allow oeconomic resource management.

3.3 Object Oriented Views





Object Oriented Views (OOV) are small, surveyable class and object models. The purpose of the concept of OOVs is to emphasize the role of such small models.

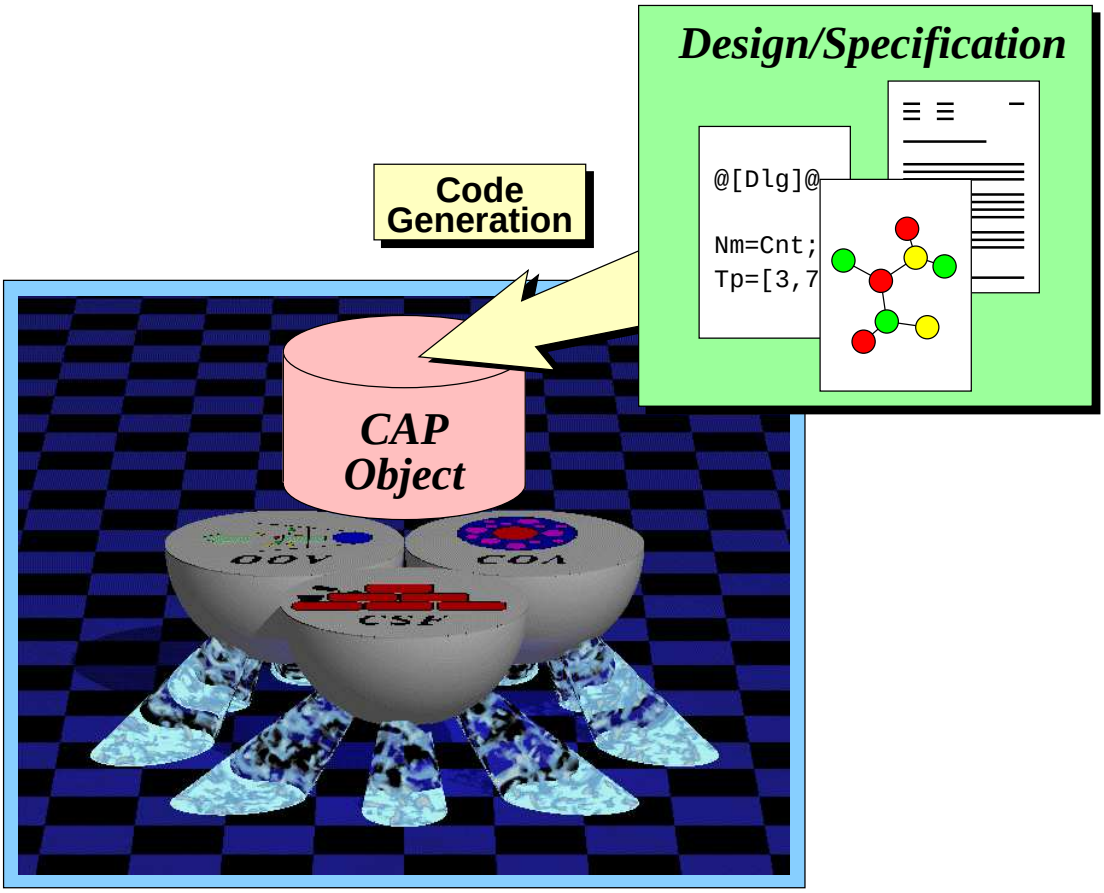
OOVs serve as the interface between subsystems and their environment. Modeling these OOV interfaces gives a precise description of the coupling of the subsystems to their environment, which is a valueable base for porting, adapting or maintaining them.

Furthermore, OOVs provide a description of a system related to a users point of view, thus incorporating *subject orientation*. Therefore they help to focus on the user's needs without making compromises in formulation, since the specification is not taken and mixed, but saved.

An object oriented system with OOVs is described by one or more "central" class models, and several OOVs.

The emphasis of the subjective role of class models and arbitrarily coupled subsystems, is in contrast to hierarchical and somewhat monolithic class models more suitable to describe complex and open systems.

4 Using the Architecture



CAP *objects are build on the components* CSF, COA *and* OOV, *a flexible, powerful and movable platform. To take full benefit of the power of* CAP, *appropriate design and specification methods and languages are necessary, embedded in work bench tools and integrated with code generators.*

CSF, COA and OOV provide a powerful platform to build high quality CAP objects on. To simplify the process of creating CAP objects and to preserve maintainability, understandability and reusage of them, appropriate design tools are necessary.

It is not the goal of CAP to introduce *"The"* CAP *Designer*, or *"The"* CAP *Language* since due to the design principle of openness CAP objects should be editable from ideally arbitrary platforms, not only from one. Naturally, this implies probably the renunciation to edit some (or many) aspects, depending on the capabilities of the respective design tool.

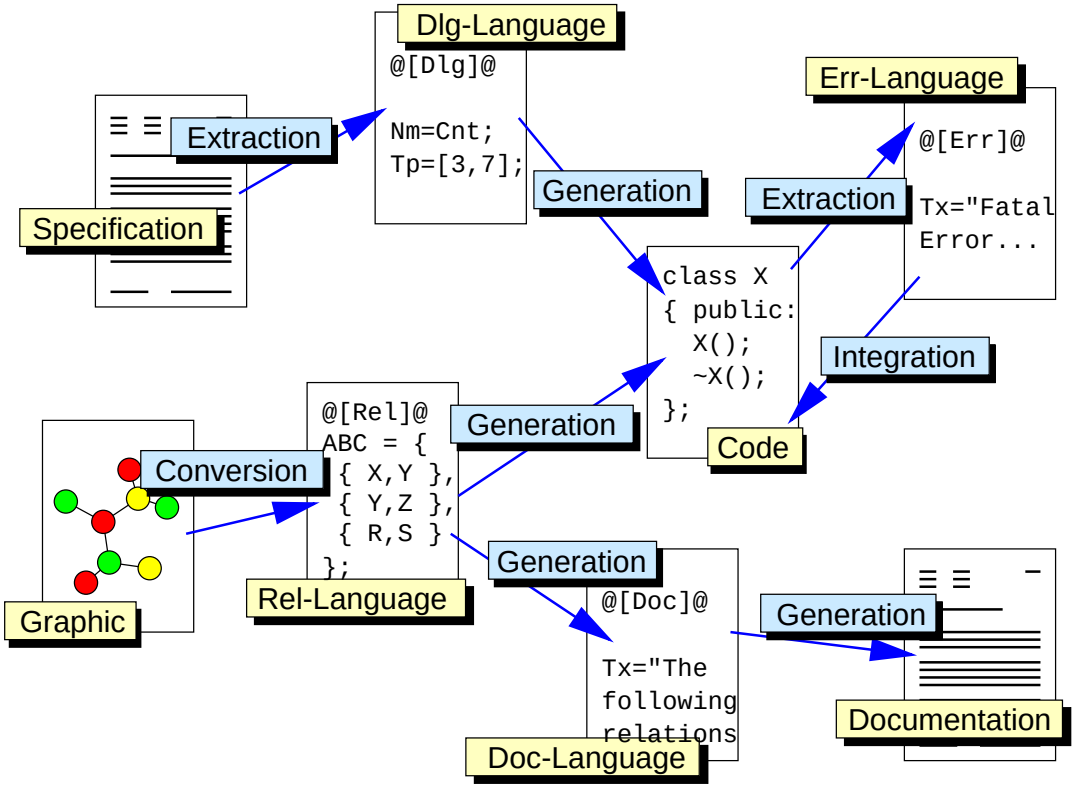
Instead, another approach is used within CAP, which is outlined in the next section.

4.1 Design and Specification

The design and specification support provided by CAP consists of an extendable set of – probably simple and small – expression capabilities like languages and graphic representations.

These capabilities may be used independently or can be combined to larger and more complete ones. There may exist several conversion tools (generalized code generators), which can translate from one kind of specification to another.

Ideally, all specification documents created are linked somehow together using the conversion tools, forming a graph whose pathes result finally in a complete implementation, as shown exemplary in the following figure.



A specification graph. The various documents are linked together with various tools, performing some kind of conversion, extraction or integration.

As a final step the various tools and appropriate editors have to be integrated into several workbenches and design environments. This system of tools and editors could be build itself on the base of CAP objects.

4.1.1 A Smart Specification Approach

To allow the easy invention of small specification languages, a prototype was already implemented which performs the following tasks:

Extraction of small specification pieces from code or text documents. These specification pieces are denoted somehow, opening a "sub-context" in their enclosing documents. E.g., in C or C++ documents they are enclosed in commentary sections, whose first content is a context introducer "`@[`*name of context*`]@` ...". The context defines a language to be used inside the context.

Code Generation based on information described in a C-style structured tag language. The definition of new code generators for new (simple) languages can be very easy performed: only a skeleton document for the target code has to be created. Mechanisms to access the parsed information are available, if necessary, all features of C++ may be used inside these skeletons.

Integration of created code into existing code or skeleton files, allowing recreation and preservation of manual changes. Generated information can be integrated in documents at arbitrary and multiple places, preserving changes at user entry points and allowing nested insertions.

These three tasks may be either performed independently or used together. If used in conjunction, the only thing to do to introduce a new language for use inside code documents and to reintegrate the result is to define the skeleton mentioned above.

5 Conclusion

The CAP is a unifying and integrating approach to put several concepts for building software together.

The project focuses on building robust, flexible, rich equipped, powerful software components and systems in today's evolving software world. Following the architectur helps to build reusable and portable components.

All in all, CAP helps to survive in the CyberSpace.