Cooperation models in co-design : application to architectural design.

G. Halin*, K. Benali**, J.C. Bignon*, C.Godart**,

* CRAI
Ecole d'Architecture de Nancy, 2 rue Bastien Lepage. 54000 Nancy, France.
Tel: (33) 3 83 30 81 46. Fax: (33) 3 83 30 81 27. Email: halin@crai.archi.fr
** LORIA
Campus Scientifique BP 239, 54506 Vandœuvre-les-Nancy, Cedex, France.
Tel: (33) 3 83 59 20 00. Fax: (33) 3 83 30 81 27.
E mail: godart@lorraine.loria.fr

ABSTRACT

This paper focuses on cooperation concepts necessary for managing concurrent engineering. It reports on a research work being done in a project which establishes a connection between computer sciences, architecture and telecommunications research[1]. Simple electronic cooperation paradigms (also called generic cooperation bricks) are found by analysing the current usage of human cooperation in the domain of AEC design environments. We introduce the principles of a middleware to build easily cooperative applications to assist cooperative design. In this approach, the design actors choose cooperation forms by instancing adapted generic cooperation bricks.

1 INTRODUCTION

This paper reports on a research work being done in a project which establishes a connection between computer sciences, architecture and telecommunications research[1]. It focuses on cooperation concepts necessary for managing concurrent engineering. This topic is present in every large building project and especially in design phase. Concurrent engineering and management of activities dispatched over a network are currently of high interest (Julien 1995) (Pawar et al. 1995). Numerous research projects are starting or restarting in conjunction with the evolution of hardware and software frameworks.

The impact of PC's architecture and Internet allows geographically distributed partners to communicate through simple interfaces. For the moment, these communications and exchanges are directed by two forms : point to point with very simple exchanges (mailing and file), or centralised where each exchange is planned and must be controlled. Our approach is located between these two forms. Indeed we

[1] This project is supported by the CNET France.

think that electronic communication does not have to change the actor habits fundamentally as it brings a significant profit or advantage which compensates this habit modification. Then, the cooperation model proposed here should be nearest from the current usage or should be the usage expected which is never put into practice.

The analysis of current usage in the domain of AEC design reveals that the cooperation is often short-lived and organised between firms of different sizes and with different organisations. From this analysis, we find out simple cooperation paradigms (we call them *generic cooperation bricks*) and we introduce the principles of a middleware to build cooperative applications. Then, a cooperative application is a combination of bricks of cooperation which are themselves instances of generic cooperation bricks. This form of cooperative applications must be safe but also easy to use for each actor.

2 ANALYSIS OF CURRENT USAGE

2.1 **General scope**

The AEC domain has many big enterprises but its major activity is done by small enterprises. Typically, the construction of a building is the result of cooperation between numerous and different actors (architects, structural engineers, mechanical engineers,...) who create and use a temporary relational system which is commonly called « Virtual Enterprise ». This kind of collaboration is in contrast to the one existing in the manufacturing industry which is strongly integrator with permanent relations between actors. This specific character gives the AEC actors a very older expertise of these cooperative practices. Paradoxically, it puts a break on the development of communication tools for EDI (Electronic Data Interchange). The usage of actor communication is anchored in the practice. The problems are often solved during oral discussion on building site. An electronic communication does not have to change this form of collaborative work, it has to be adapted to this form and propose simple protocols of exchange.

A first level of collaboration concerns data exchange. Many projects (Björk 1995) (Tonarelli et al. 1997) (Junge and Liebich 1997) propose a product data model which contains all information needed during the building life cycle. A great effort is done by the IAI (International Alliance of Interoperability) which proposes the IFCs (Industry Foundation Classes) with a bottom up and incremental approach (IAI 1996). The intention of the IAI is to define a specification of all the objects (IFC) that could appear in a building project. Parallel to these projects, stands the STEP project (STEP 1994) with the objective to build a broad standardisation of product data exchanges. The STEP project uses a top down approach and will soon integrate the IFCs in its model proposition.

This first level must produce CAD applications which will be able to communicate standard product information. This is important to realise the second level of collaboration where the cooperative work is done in the framework of a Virtual Enterprise.

At the moment, this second level is dominated by two forms of collaboration :

- the first form is a simple data exchange with floppy disk or electronic mail. This approach is close to the current usage of plan and administrative document exchanges between actors,
- the second form is the centralised approach which is only effective in important projects. This form of collaborative work, which is either called « cell of synthesis » or « electronic store of plans», is expensive for small enterprises. Indeed, it requires specific equipments and specific human investments. The most important disadvantage of such approach is that it changes actor habits while adding new actors for control exchange. Thus, this current form of centralised collaboration is not adapted for medium or small building projects.

Between these two forms of collaboration, the new computer technologies, as Internet, CORBA (OMG 1995) and JAVA (Atkinson M.P. et al. 1996), allow to envisage a new approach of cooperative work where design activities are distributed on a network. This new form of collaboration can be viewed with different scenarii depending on the project size. The VEGA Esprit project (Junge R. 1997) tries to represent the entire workflow process which will be able to control the project coherence during the actors' distributed activities. This approach is adapted for projects with an important investment.

Our approach considers a project as a set of specific collaborations which can be used according to needs encountered during the design process. These short time collaborations are the components of what we call a « Project Enterprise ».

For example, instead of waiting the end of a specific design task to transmit the result to a co-designer, it seems more interesting to provide a working copy of the design to allow him to react and enhance the design in progress.

The general objective of our study is to find cooperative behaviours in the current usage of collaboration, in order to specify generic bricks of cooperation and realise cooperation tools which will be well-received and used by the AEC actors. This study will consider more specifically collaborations in small or medium projects.

2.2 **Case Study**

To illustrate our approach, we reuse the example developed in (Rosenman and Gero 1996). It consists in designing a one-storey apartment containing a living room with a glass wall. Three kind of designers cooperate to achieve this work: the architect providing spatial organisation, the structural engineer in charge of building stability, and the HVAC engineer responsible of the thermal behaviour of the apartment. These activitites are done with a sequential workflow management.

The architect's activity is to design and represent the apartment spatial organisation with walls, windows,... Such a representation may lead to the plan of figure 1. This graphical representation only takes care of volumes, spaces and luminosity of the apartment. We limit this example to the living room.

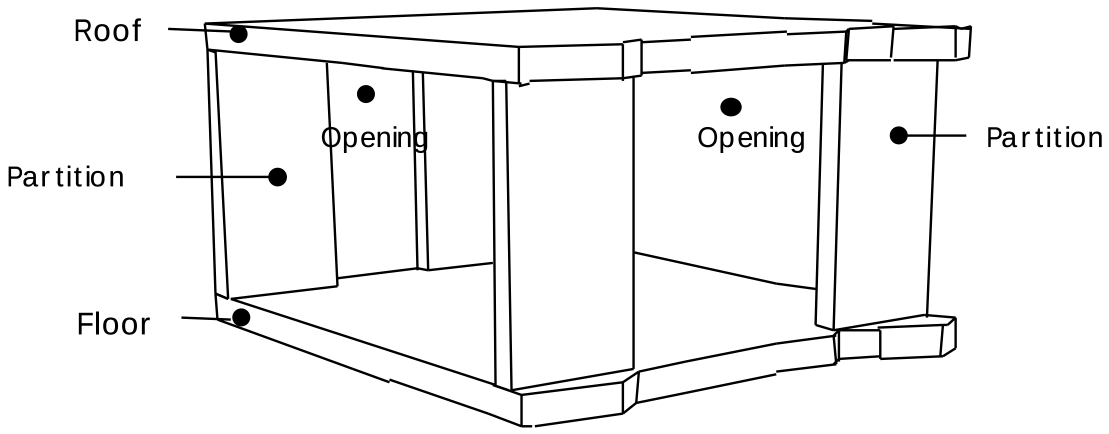

Figure 1 : **Apartment spatial organisation.**

The design activity of the architect can also be formalised by a schedule which organises a sequence of operations (see figure 2). This activity is called A_0. The architect operations are noted :

- $plan_0$ = edit() when the architect creates a new plan,
- $plan_0$ = edit($plan_0$) when the architect updates the plan,
- write (plan,$plan_0$) when the architect provides the plan to the other activities by storing it in the drawing archive storage.

In the example below (cf. figure 2), the architect constructs his plan in three times before storing it.

A_0: Produce plan
$plan_0$=edit()
$plan_0$=edit($plan_0$)
$plan_0$=edit($plan_0$)
write(plan,$plan_0$)

Figure 2 : **Architect's activity.**

The structural engineer activity consists in specifying the structural elements of the apartment. Such elements (cross walls , beams,...) will be chosen to respect, as far as possible, the choices made by the architect and the overall harmony of the building. Such an activity can lead to the modified plan depicted in figure 3. In this new plan, the structural elements corresponding to the architect plan are highlighted: cross walls chosen among architect separation walls, "transforming" floor and roof into structurally specified slabs. And new structural elements are added to the plan: addition of a beam which has its structural specification defined. This description only takes care of structural aspects of the apartment.

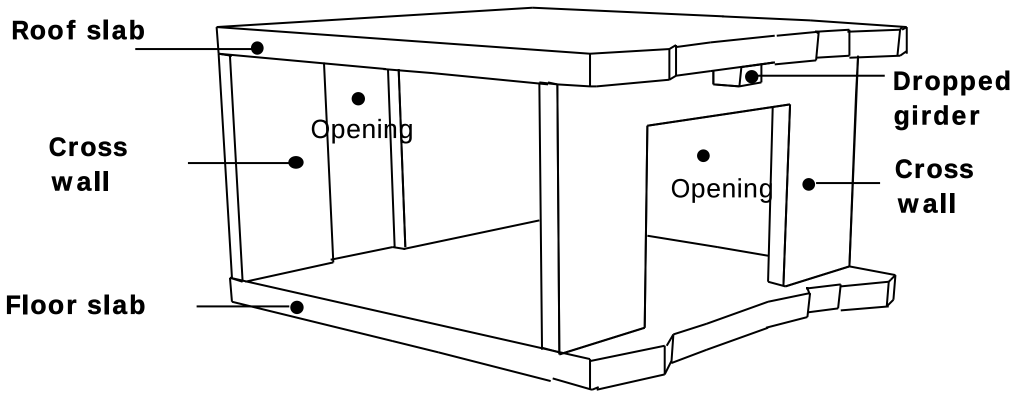

Figure 3 : **Apartment with structural elements.**

The results of the structural engineer design can be formalised as in figure 4. A_1 is the activity of the structural engineer. His operations are noted :

- $plan_1$ <- read(plan) when the structural engineer retrieves the plan from the drawing archive storage,
- $plan_1$ = edit($plan_1$) when the structural engineer updates the plan,
- write (plan,$plan_1$) when the structural engineer provides the plan to the other activities by storing it in the drawing archive storage.

In the example below (cf. figure 4), the structural engineer adds structural elements to the plan in two times before storing it in the drawing archive storage.

A_1: Produce plan with structures

$plan_1 \leftarrow$ read(plan

$plan_1$=edit($plan_1$)

$plan_1$=edit($plan_1$)

write(plan,$plan_1$)}

Figure 4 **Structural engineer's activity.**

At the end of the structural engineer work, the architect retrieves the plan for controlling that modifications respect the overall consistency of the project. In particular, the addition of the beam will cause modification to the storey height. According to this new situation, the architect may decide to modify the size of the glass wall (or eventually to move it). Another element to take into account is the new thickness of cross walls which may differ from the separation wall initial thickness. The architect makes a synthesis of his vision and of the structural engineer's one to produce a complete plan including structural elements. He does it in one step as shown in figure 5. The new plan is depicted in figure 6.

A_0: Produce new plan with structures
$plan_0 \leftarrow$ re-read(plan)
$plan_0$=edit($plan_0$)
write(plan,$plan_0$)

Figure 5 : **Architect synthesis activity.**

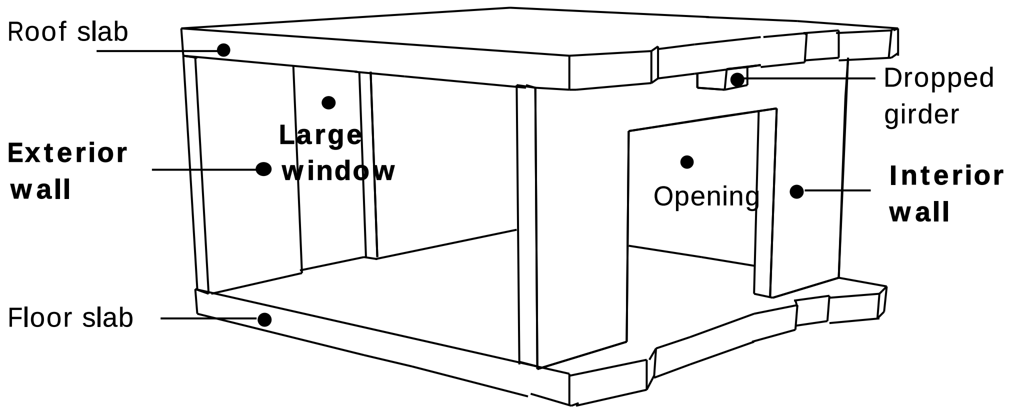

Figure 6 : **Apartment after architect synthesis activity.**

Finally, to achieve architectural design, the HVAC engineer will intervene to change the glass wall according to the climate and the apartment exposure. This choice will be guided by various considerations issued from the plan of figure 7. His activity can be summarised as in figure 8.

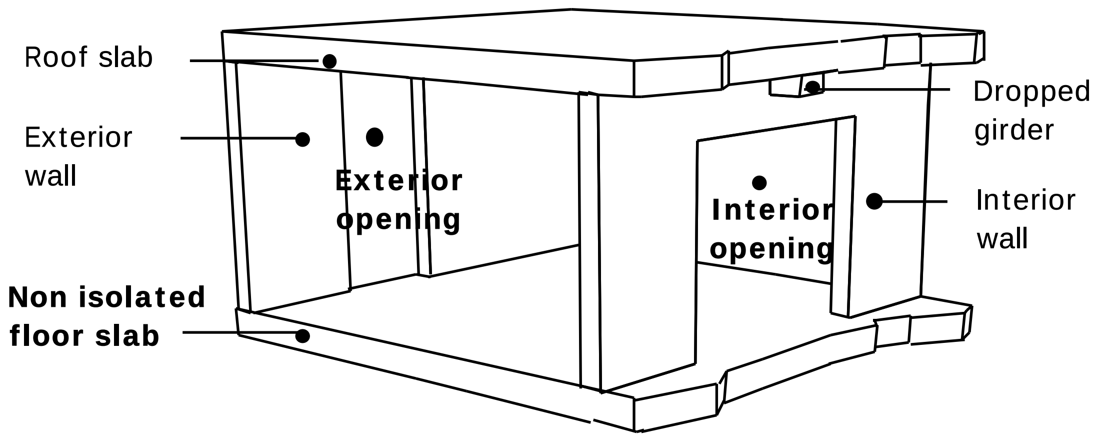

Figure 7 : **Apartment after the HVAC engineer's activity.**

A_2: **Produce specification of the wall glass**
$plan_2 \leftarrow$ read(plan)
$wallglass_2$=edit()
write(wallglass,$wallglass_2$)

Figure 8 : **HVAC engineer's activity.**

3 SUPPORTING COOPERATION.

3.1 **Generic bricks of cooperation.**

In the example developed above, we presented the activities of the various actors as executed in sequence, each actor storing in the drawing archive storage a finished work, and each actor waiting until the end of another actor work. In this section, we show how these actors can take advantage of interactions during their individual work. That is well known, but our contribution is to show how we can formalise this cooperation and develop specific softwares to support it (Benali et al. 1998).

3.1.1 *The Client/Server paradigm*

With the organisation described above, the HVAC engineer does nothing before having obtained the definitive plan with structural informations and the modified wall glass. Nevertheless, even if the final result of the HVAC engineer has to consider the final plan with structural informations, the HVAC engineer is able to start his work on the initial architect's plan. Effectively, this plan already has a wall glass, and thermal work can start on this basis. The apartment exposure and the huge size of the wall glass provide elements to start work. This kind of cooperation corresponds to a paradigm of cooperation which is often repeated. We call it the Client/Server paradigm. In this paradigm, one actor (the client) can work on preliminary versions produced by another actor (the server). The only compulsory rules are : the server must produce the final version of each object it has produced in a preliminary version, and the client take in account at least the last version produced by the server. This paradigm allows some cooperative work which enhances productivity and allows to start HVAC engineer activity A_2 before the end of architect activity A_1. To implement this paradigm, the architect has to store in the drawing archive storage, different preliminary versions of his plan. Figure 9 presents an example of such a cooperation. between the architect and the HVAC engineer.

A_0: Produce plan	**A_2: Produce specification of the wall glass**
$plan_0$=edit()	
$plan_0$=edit($plan_0$)	
write (plan,$plan_0$)	
	$plan_2 \leftarrow$ read(plan)
$plan_0$=edit($plan_0$)	$wallglass_2$=edit()
write (plan,$plan_0$)	
	$plan_2 \leftarrow$ read(plan)
	write(wallglass,$wallglass_2$)

Figure 9 : **Architect and HVAC engineer cooperative work.**

In the example above, the architect is the only one who updates the plan; the HVAC engineer does only read the plan for producing another objet (the wall glass specification).

3.1.2 *The Cooperative Write paradigm*

The collaborative process between the architect and the structural engineer corresponds to another kind of cooperation. In fact, both of them update the plan. We developed the example in a sequential way : at one step only one actor updates the plan. The architect creates the plan, then the structural engineer adds informations, and finally the architect synthesises the work and produces the final plan. This kind of work organisation does not allow to take profit from possible synergy between two actors really cooperating. This is not the better way to do things : it is perfectly possible to allow real cooperation between the two actors from the beginning of the plan elaboration. The structural engineer can start his activity as soon as possible and then provide the architect a more accurate vision of the actual volumes of the plan final version. To do that, the two actors must be allowed to write at the « same time » a common plan.

In our example, the height of the wall glass is imposed by the structural engineer. Cooperative work would allow these two actors to agree more quickly on a common view. We have pointed out this behaviour as being really common : it defines what we call the *cooperative write* paradigm in which two actors modify at the same time the same object. They have to follow some rules : to be aware of each other work and to converge towards a same view of this object. Figure 10 depicts a schedule of such a cooperation.

A_0: Produce plan	**A_1: Produce plan**
$plan_0$=edit()	
$plan_0$=edit($plan_0$)	
write(plan,$plan_0$)	
	$plan_1 \leftarrow$ read (plan)
$plan_0$=edit($plan_0$)	
write(plan,$plan_0$)	
	$plan_1 \leftarrow$ re-read(plan)
$plan_0$=edit($plan_0$)	$plan_1$=edit($plan_1$)
	write(plan,$plan_1$)
$plan_0' \leftarrow$ re-read(plan)	
$plan_0$=merge($plan_0$,$plan_0'$)	
write(plan,$plan_0$)	
write(plan,$plan_0$)	
	$plan_1 \leftarrow$ re-read(plan)

Figure 10 : **Architect and structural engineer cooperative writing.**

At this point, we have introduced two modes of cooperation. In the first one, an actor uses the plan produced by another to design the object wall glass for which he is responsible (client/server paradigm). In the second, the two actors cooperate to produce one object (one plan). Lets us now consider a third one called *writer/reviewer.*

3.1.3 *The writer/reviewer paradigm.*

This third form of cooperation corresponds to the case in which an actor produces an object under the control of another. As an example, an architect may be controlled by a town planner which gives advices in return and which validates the final plan. In this mode of cooperation, the writer produces different successive versions of the object it has to produce, especially to take into account the review of the town planner. This review has for objective to impose the architect to respect some rules. As example, he can react on the first version of the plan and ask for the surface of the wall glass to be reduced or the type of the woodwork to be changed. On a following version, he can react on the global look of the building. We say that the interactions between the architect and the town planner follow the *writer/reviewer* paradigm. The writer is the architect, the reviewer is the town planner. This mode of cooperation is illustrated in figure 11.

At this point, we have introduced the different paradigms of cooperation and we have illustrated them through examples. These examples have the peculiarity that they depict schedules in which things run well (the interactions between partners follow the cooperation protocol). The activities will be validated and their modifications will impact the object base (or the drawing archive storage in other terms). But what we want to underline now is that the interest of our approach consists in preventing actors from violating the process of cooperation: rules about object sharing prevent an actor to terminate its work if the current state of this work violates a rule of the current process .

A_0: Produce plan	**A_1: Write report**
$plan_0$=edit()	
write(plan,$plan_0$)	
$plan_0$=edit($plan_0$)	$plan_1 \leftarrow$ read(plan)
	rep_1=edit()
	write(rep,rep_1)
$rep_0 \leftarrow$ read(rep)	
$plan_0$=edit($plan_0$)	
write(plan,$plan_0$)	
	$plan_1 \leftarrow$ re-read(plan)
	rep_1=edit(rep_1)
	write(rep,rep_1)
$rep_0 \leftarrow$ re-read(rep)	

Figure 11 : **Writer/Reviewer.**

This achitecture of collaboration between the project actors is resumed in the figure 12.

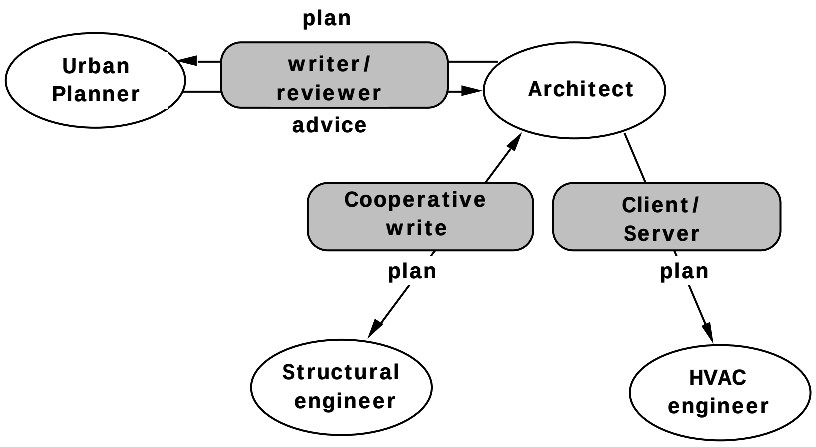

Figure 12 : **Cooperation architecture.**

3.2 Combining cooperation bricks.

3.2.1 *The problem*

In the previous section, we have individually studied different cooperation paradigms. In a real cooperative application, different instances of the same paradigm, applied to differents objects and/or different activities, and different instances of different paradigms, live together. Each instance defines a brick of cooperation and a cooperative process is built by combining several bricks. With this view of things, the idea of a generic brick corresponds to this of cooperation paradigm as defined before. The problem we address in this section is: how to define a cooperation brick and how to combine these bricks ?

This problem is quite complex. In the *writer/reviewer* scheme, a designer allows a partner to have knowledge of an object being elaborated (before it is stabilised). This allows the second to react quickly and to enrich the current design. Generally, such exchanges repeat and define a cross fertilisation loop. The control of such a loop is not so easy. In the *cooperative write* scheme, two partners work at the same time on the same object; we can easily imagine the synchronisation problem that this implies. And now, suppose that an object written in such a cooperative write participates to a *writer/reviewer* scheme with a third partner !

The question is how to control these flows of information to ensure consistency of the results in a way as much as possible transparent to the professionals in front of their screen ? We have considered that our problem can be compared in some ways to the problem of concurrency control in database, when several activities access the same object of the database at the same time, either to read it or to write it.

3.2.2 *A Transactional Approach*

To support the concurrency control in traditional applications, database systems implement the concept of a transaction. A transaction is an execution of a program segment that performs some functions or activities by accessing and manipulating a shared database. Deposit, withdrawal, and transfer of money are typical examples of business transactions whereas automative engine design or the debugging of a software are examples of CAD/CASE transactions.

A transaction model assumes that if each activity works correctly individually and if each such correct activity is encapsulated in a transaction, the parallel execution of a set of concurrent activities is also correct. In other words, if each activity takes the database from one consistent state to another, the parallel execution of a set of such activities encapsulated in transactions takes also the database from one consistent state to another. To assert correctness, a transaction model implements a synchronisation protocol which imposes rules to activities on the intertwining of their read and write operations. Such a protocol makes references to a correctness criteria.

The interest of this approach is that it requests no, or rather few, programming specific to the problem of interactions between activities.

Typically, the more popular protocol is the *Two Phases Locking* which refers to the *Serializability* criteria (a parallel execution of a set of activities is correct if it is equivalent to a serial execution of these activities) (Bernstein P.A. et al. 1987). This protocol is well adapted for traditional business processes but not for CAD applications. And this is the case for most traditional protocols developed for traditional database applications. However, during the last decade important advances have been made in concurrency control and transaction processing, especially to manage new characteristics of the new applications: long duration, interacting, cooperative, ... (Elmagarmid A. K. et al. 1992) Our approach contributes to these advances.

Broadly speaking, the main idea developed in the new techniques is to exploit semantics of data items and operations defined on them, the behavioural properties of the activities in the database, and the correctness requirements of applications to surpass the limits of serializability. These new techniques are referred as semantics-based transaction processing techniques. They can be classified in two sub-approaches:

· the first consists in founding by default the correctness on serializability, but to relax it when requested by taking into account some knowledge on the activities being synchronised,
· the second consists in defining new correctness criteria which are more related to new application requirements and new protocols which enforce these new criteria.

Our approach enters in the second scheme: especially, we have defined a new correctness criterion which defines a large sphere of security in which activities can cooperate in the sense of the three cooperation paradigms defined above. Then semantics of the application can be used to restrict the possibilities of cooperation.

3.2.3 *Our protocol to support cooperation*

To execute, any activity is encapsulated in a long term transaction. When executing, an activity can publish at any time intermediate results, which are potentially inconsistent and non stable (by finition). Publishing intermediate results is the main way to cooperate in our environment. Applied to our illustrative example, we have a first transaction for the architect, a second for the structural engineer and a third for the HVAC engineer. The preliminary versions of plans are the intermediate results.

The following set of nine rules define our protocol (called the COO protocol) (Molli P. 1997). The first set of four rules is sufficient to synchronise one individual client/serve brick. The last five rules are requested to synchronise cooperative write and writer/reviewer bricks.

1. A result produced before the end of a transaction is an intermediate result. A user can produce at any time an intermediate result.
2. A result produced at the end of a transaction is a final result. If a transaction produces an intermediate result, then it must produce the corresponding final result. A transaction keeps track of all the intermediate results it has produced and produces automatically the corresponding final results during its termination phase.
3. If a transaction reads an intermediate result of another transaction, then it must read the corresponding final result. If a transaction reads an intermediate result of an object produced by another transaction, then a dependency is created between the two transactions. This dependency is removed when the depending transaction reads the final value of the corresponding object.
4. If a transaction tries to terminate and at least one dependency exists between itself and another transaction, the transaction cannot terminate.
5. The transactions which are implicated in a cycle of dependencies define a group of transactions and must terminate simultaneously. This is especially the case of two transactions which cooperate in a cooperative write brick or in a writer/reviewer brick.
6. A transaction starts a group termination by trying to terminate itself. By this action, it produces a set of potentially final results and becomes « ready to terminate ».
7. If another transaction of the same group tries to commit and all others transactions of the group are in a «ready to terminate » state, then all transactions of the group terminate simultaneously.
8. If another transaction of the same group tries to terminate and all others transactions of the group are not in the « ready to terminate » state, then it completes the set of potentially final results and becomes « ready to terminate ».
9. If a transaction of a group produces a new intermediate result, then the group termination tentative is aborted and all transactions of the group become again « active ». This is the way for a transaction to show its disagreement with the potentially new state of the shared objects.

4 IMPLEMENTATION

A first version of this protocol has been implemented on top of an object oriented database management system called P-ROOT (Godart C. et al. 1996). A second version is being developed in JAVA (Atkinson M.P. et al. 1996) with the objective to respect two main constraints :

1. the environment must support cooperation without changing the habits of designers. Especially, they must continue to work with their tools and their usages,
2. it must run on a simple computer infrastructure, typically a network of personal computers.

AEC applications will be used to validate the prototype: the project in which the prototype is being developed implicates researchers of three different domains: computer sciences, architecture and telecommunications.

5 CONCLUSION

This paper has formalised some principles of cooperation, taking advantages of the experience developed in several studies about co-design, and especially in the domain of AEC design.

Our cooperation model is based on three simple generic bricks that we are able to instanciate and to combine in a safe way. If this set of basic generic bricks seem small, we have experimented that it responds to a large set of cooperation cases and appears to be very useful. However, one important objective of our current work is to find out new basic generic bricks of cooperation and to integrate them in a theorical model. This theorical model must allow the development of software which should be easy to use by professionals.

6 REFERENCES

Atkinson, M.P. Jordan, M.J. Daynés, L. Spence, S. (1996) Design issues for persistent Java : a type-safe, object-oriented, orthogonally persistent system. *In POS-7,* 1996.

Benali, K. Munier, M. Godart, C. Cooperation models in co-design. *In: International Conference on Agile Manufactoring.* Minneapolis, june 1998.

Bernstein, P.A. Hadzilacos, V. Goodman N. (1987) Concurrency control and recovery in database systems. *Addison-Wesly*, 1987.

Björk, B.C. (1995) Requirements and information structures for building product data models, *Espoo : Technical Research Center of Finland (VTT),* 1995 , 163 p. VTT Publication n°245.

Elmagarmid A.K. et al. (1992) Database transaction models for advanced applications. *Morgan Kauffman*, 1992.

Godart, C. et al. (1996) Designing and implementing COO. *In International Conference on Software Process (ICSE18)*. IEEE Press, 1996.

IAI (1996) End User Guide to Industry Foundation Classes, *International Alliance of Interoperability*, 1996.

Julien, P.A. (1995) « Wide-Entreprise » : contraints et opportunities, *Proceedings of Concurrent Engineering & Technical Information Processing, ILCE'95,* Paris, January 1995, pp 27-38.

Junge, R. Köthe, M. Schulz, K. Zarli, A. Bakkeren, W. (1997) The VEGA platform. IT for the Virtual Enterprise, *Proceedings of CAAD Futures,* 1997 , pp 591-616.

Junge, R. Liebich, T. (1997) Product data model for interoperability in an distributed environment, *Proceedings of CAAD Futures,* 1997 , pp 571-589.

Molli, P. (1997) COO transactions: enhancing long transaction model with cooperation. *In 7th Software Configuration Management Workshop (SCM7),* LNCS, 1997.

Object Management Group (1995) CORBA Sevices : Common Object Services Specification. *Revised Edition, 95-3-31*, 1995.

Paward, K.S. Thoben, K.D. and Oehlmann R. (1995) A Holistic infrastructure for the guided implementation of concurrent engineering principles. *Proceedings of Concurrent Engineering & Technical Information Processing, ILCE'95,* Paris, January 1995, pp 51-62.

Rosenman, M.A. Gero, J.S. (1996) Modelling multiple views of design objects in a collaborative CAD environment. *Computer-Aided Design*, 28(3), pp193--205, 1996.

STEP (1994) Overview and Fundamental Principles, *ISO-10303, part 1,*1994.

Tonarelli, P. Ferriès, B. Delaporte, J.L. Tahon, C.(1997) Proposal of a product model for the buildong trade, *Automation in Construction, vol 5, pp 501-520.*