ABSTRACT
In real-time collaborative groupware, shared objects are replicated on distant geographical sites. Each user works on his own copies. This implies the divergence of different copies. Operational transformation approach makes it possible to reconcile the divergent copies. It allows to ensure the syntactic consistency of the copies by ensuring: convergence, causality and user intention. Although these properties resolves the problems involved in the conflicting accesses on the copies, they do not ensure the semantic consistency of the copies with respect to the context of the application. To solve this problem we propose to integrate semantic constraints to the operational transformation approach. This article represents the constraints integration.

Keywords
collaborative work, operational transformation, semantic consistency, constraints.

INTRODUCTION
In real-time collaborative groupware, shared objects are replicated on distant geographical sites. Operational transformation approach ensures the convergence of the copies. The principle of this approach is to integrate concurrent operations on the various sites to reach a state of convergence while respecting the causality and the intention of the operations [1,2].

The convergence of data on different copies does not necessarily mean a consistent state. But by accounting for the semantics of the application, we can reach a convergent and a consistent state.

A common way to ensure semantic consistency is to define a set of constraints and to check these constraints. In order to integrate constraints in a system, it is necessary to answer the following questions:

1. what is the language of description of the constraints ?
2. Where are the constraints imposed? on each site, in a central site?
3. When are they checked? and how?
4. What measures to take in case the constraints are violated?

Moreover, in the context of replicated data, the consistency of the data not only depends on the local operations but also on the operations taken on the other copies. So it is necessary to determine the impact of semantic constraints on the operational transformation approach and vice-versa.

In this paper, we will answer these questions. We start with a general representation of the operational transformation approach. Thereafter, we present the language of definition of the constraints, how to check them and when. Section 4, shows the implementation of the constraints in the SAMS environment[4]. This environment implements the operational transformation algorithm SOCT4 [3]. Section 5 compares our proposition with related works. The last section concludes the paper and points future works.

TRANSFORMATIONAL APPROACH
Operational transformation approach was originally used in real-time groupware systems [1,2,3] to ensure consistency of copies of shared data. The model of transformational approach considers \( n \) sites. Each site has a copy of the shared objects. When an object is modified on one site, the operation is executed immediately and sent to other sites to be executed again. So every operation is processed in four steps:

1. generation on one site,
2. broadcast to other sites,
3. reception by other sites,
4. execution on other sites.

The execution context of a received operation \( op \) may be different from its generation context. In this case, the integration of \( op \), by other sites may lead to inconsistencies between replicas. We illustrate this behavior in figure 1. There are two sites \( site_1 \) and \( site_2 \) working on a shared data of type \( String \). We consider that a \( String \) object can be modified with the operation \( Ins(p, c) \) for inserting a character \( c \) at position \( p \) in the string. We suppose the position of the first character in string is 0. \( user_1 \) and \( user_2 \) generate two concurrent operations: \( op_1 = Ins(2, f) \) and \( op_2 = Ins(5, s) \). When \( op_1 \) is received and executed on \( site_2 \), it produces the expected string "effects". But, when \( op_2 \) is received on \( site_1 \), it does not take into account that \( op_1 \) has been executed before it. So, we obtain a divergence between \( site_1 \) and \( site_2 \).
In the operational transformation approach, received operations are transformed according to local concurrent operations and then executed. This transformation is done by calling transformation functions. A transformation function $T$ takes two concurrent operations $o_1$ and $o_2$ defined on the same state $s$ and returns $o'_1$. $o'_1$ is equivalent to $o_1$ but defined on a state where $o_2$ has been applied. We illustrate the effect of a transformation function in figure 2. When $o_2$ is received on site 1, $o_2$ needs to be transformed according to $o_1$. The integration algorithm calls the transformation function as follows:

$$T(\text{Ins}(5, s), \text{Ins}(2, f)) = \text{Ins}(6, s)$$

The insertion position of $o_2$ is incremented because $o_2$ has inserted $f$ after $s$ in state “effect”. Next, $o'_2$ is executed on site 2. In the same way, when $o_1$ is received on site 2, the transformation algorithm calls:

$$T(\text{Ins}(2, f), \text{Ins}(5, s)) = \text{Ins}(2, f)$$

The example makes it clear that the transformational approach defines two main components: the integration algorithm and the transformation functions. The Integration algorithm is responsible for receiving, broadcasting and executing operations. It is independent of the type of shared data, it calls transformation functions when needed. The transformation functions are responsible for merging two concurrent operations defined on the same state. They are specific to the type of shared data (String in our example). A more theoretical model is defined in [1,2,3]. To be correct, an integration algorithm has to ensure three general properties:

**Convergence** When the system is idle (no operation in pipes), all copies are identical.

**Causality** If on one site, an operation $o_2$ has been executed after $o_1$, then $o_2$ must be executed after $o_1$ in all sites.

**Intention preservation** If an operation $o_i$ has to be transformed into $o'_i$, then the effects of $o'_i$ have to be equivalent to $o_i$.

To ensure these properties, it has been proved [2,3] that the underlying transformation functions must satisfy two conditions.

Although many operational transformation algorithms have been defined, none of them takes in consideration the semantic consistency of the objects. It is widely recognize that consistency has two dimensions [5,6,7] syntactical dimension and semantic one.

Operational transformation approach ensures syntactic consistency, which means that all sites see the same view of the data, even if that view does not necessarily make sense in the context of the application. In order to ensure consistency, the two dimensions must be taken in consideration.

The semantic of an application can be integrated directly in the operations or defined declaratively by a set of rules. The first solution does not require a language to express the rules nor an algorithm to check them. However, it is not practical in the context of the transformational approach. Each time we need to define a new constraint, we have to rewrite the operation(s), and by consequence to rewrite the corresponding transformation functions and to prove their correctness[8]. The second solution is better fitting to the context of transformational approach.
INTEGRITY CONSTRAINTS IN TRANSFORMATIONAL APPROACH

We must firstly precise which kind of constraints to add. A major distinction in the expressive power of constraints concerns their static or dynamic nature. Static constraints specify conditions that must be satisfied by the state of objects at each moment\(^9\). For example, “Two different objects cannot have the same name”. Intuitively, this constraint describes the fact that at each moment I cannot find two objects with the same name, at any site. Dynamic constraints allow expressing conditions over time-ordered sequences of states\(^4,10\), constraints on the states transition and on state sequences. We are still exploring the possibilities in specifying the dynamic constraints and will focus only on the static constraints in this paper.

Constraints Definition Language
We use first order logic for describing static constraints. More precisely the language developed in xlinkit \(^{11}\) (www.xlinkit.com) Of course, we can use other languages to define constraints \(^{12,13,14}\). This is not a critical point, since our main objective is to ensure semantic consistency in transformational approach and not to define a new language for constraints. The choice of xlinkit is based on two facts. Firstly, it defines new semantics for first order logic, which returns hyperlinks between inconsistent elements instead of boolean values. This is more valuable for inconsistency diagnosis than only boolean value. Secondly, there is an open source implementation of the constraints checking algorithm, which can be easily integrated with our SAMS environment.

Example of constraints definition
It is quite simple to use xlinkit. It takes as a set of distributed XML documents \(^{14}\) and a set of potentially distributed rules written in XML. The rules express consistency constraints across the documents. xlinkit returns a set of hyperlinks, in the form of a linkbase, which supports navigation between inconsistent elements of the XML resources. For example, if we want to define constraints on XML document which contains CRC cards \(^{15}\) (Class, Responsibility, Collaboration; CRC cards are used in objects oriented design to define classes and components of a software system).

This card can be represented by XML tree. This tree is provided with the following elementary operations:

1. CreateNode\((n, tn)\):nid\( n \) is the path to the parent node in the XML tree. \( tn \) is the tag name of the node. \( nid \) is the path to the newly created node.
2. DeleteNode\((n)\):void \( n \) is the path to the node to delete
3. CreateAttribute\((n, a, v)\):void \( n \) is the path to the node to update. \( a \) is the name of an attribute. \( v \) is the value of the attribute.
4. DeleteAttribute\((n, a)\):void deletes attribute \( a \) from the node identified by path \( n \).
5. ChangeAttribute\((n, a, v)\):void for a node identified by path \( n \), it changes the value of attribute \( a \) to a new value \( v \).

When a user creates a card, he generates a sequence of elementary operations. These operations are carried out immediately on his site. For example, the creation of the card illustrated in figure 3 generates the following sequence of operations:

CreateNode\("/\"Card\"") \(-\rightarrow\) “/0.doc.1”
CreateNode\("/0/doc.1/\"Class\"") \(-\rightarrow\) “/0.doc.1/doc.2”
CreateNode\("/0/doc.1/\"Responsibility\"") \(-\rightarrow\) “/0.doc.1/doc.3”
CreateNode\("/0/doc.1/\"Collaborations\"") \(-\rightarrow\) “/0/doc.1/doc.4”
CreateAttribute\("/0.doc.1/doc.2/\"name\",\"Model\"")
CreateAttribute\("/0/doc.1/doc.3/\"description\",\"Provides functional core of the Application- Notify dependent component about data\"")
CreateAttribute\("/0/doc.1/doc.4/\"description\",\"View-Controller\"")

The document “doc.xml” defines three CRC card:

```xml
<ROOT ID="/"
  
  <card ID="/0/doc.1/"
    
    <class ID="/0/doc.1/doc.2/" name="Model"/>
    <responsibility ID="/0/doc.1/doc.3/"
    
    <collaboration ID="/0/doc.1/doc.4/"/>
  
  </card>

  <card ID="/0/doc.5/"
    
    <class ID="/0/doc.5/doc.6/" name="Model"/>
    <responsibility ID="/0/doc.5/doc.7/"
    
    <collaboration ID="/0/doc.5/doc.8/"/>
  
  </card>

  <card ID="/0/doc.9/"
    
    <class ID="/0/doc.9/doc.10/" name="Process"/>
    <responsibility ID="/0/doc.9/doc.11/"
    
    <collaboration ID="/0/doc.9/doc.12/"/>
  
  </card>
```

Figure 3: CRC Card
The CRC card above has "Model" as name, two responsibilities and it collaborates with two other classes.
This document must satisfy the constraint: "two distinct CRCcard cannot have the same name". We can enforce this constraint by defining the operation “CreateCRCCard(name,definition)” and handle the name conflict in the transformation function T(CreateCRCCard,CreateCRCCard). But defining new operations requires to define new transformation functions and to prove the correctness. This task is quite complex and time consuming. We can enforce this constraint with operations defined at the XML level of granularity by adding an xlinkit constraint:

```
<consistencyrule id="r1">
  <header>
    <description>
      The name of a CRCcard is unique
    </description>
  </header>
  <linkgeneration>
    <consistent status="off"/>
    <inconsistent status="on"/>
    <eliminatesymmetry status="on"/>
  </linkgeneration>
  <forall var="a" in="/ROOT/card">
    <forall var="p" in="/ROOT/card">
      <implies>
        <equal op1="$a/class/@name" op2="$p/class/@name"/>
      </implies>
    </forall>
  </forall>
</consistencyrule>
```

Figure 4: The XML document doc.xml

We can consider constraints as shared objects replicated at the local storage of each site. This implies that we must define transformation functions concerning constraints operations. Adding constraints means adding a new object type in the system. We call this special object Constraint. We also define two new operations:

1. CreateNodeConstraint(n, tn)
2. CreateAttributeConstraint(a)

Now, it is necessary to define the transformation functions for these new operations. We present an example of transformation using the notations used in section 2. The transformational function T takes as parameter a distant operation and a concurrent local one. The result of the transformation is a new operation. The following function defines how to transform a CreateNodeConstraint operation by considering that the operation CreateAttributeConstraint was carried out locally.

```
T(CreateNodeConstraint(n1, t1), CreateAttributeConstraint(n2))
```

```
return CreateNodeConstraint (n1, t1)
```

**When to verify the constraints?**

Now, we have our constraints defined at each site. The question is: when to check these constraints? In databases applications, it is easy to determine when to check the constraints. Constraints are checked at the end of each transaction (at the commit time). During transaction execution constraints are relaxed i.e. they can be violated. In our system, we do not have transactions. We enumerate some events, which can mark the checking of these constraints.

**At the end of each local operation**

Checking constraints after each elementary local operation is not an efficient solution. On the one hand, it will slow down the system that could not guarantee to the users any more a minimum response time. In addition, in certain cases, checking the constraints after each elementary operation has no sense. Let us take again the example of the creation of a CRCcard. The constraint defined by r1 will be checked 6 times!! To overcome this problem, we introduce the concept of macro operation.

**At the End of Macro Operations**

A macro operation is a sequence of elementary operations. Constraints are checked at the end of the execution of macro operations. For example for the CRCcard, we can add a macro operation CreateCRCCard that is a sequence of 6 elementary operations as we have seen before.

**After Integration of Remote Macro Operations**

Checking constraints at the end of a macro operation allows to ensure the consistency of the object locally at each site. It is not sufficient to ensure the global consistency. Let us consider two users user1 and user2 cooperating for the design of a new software. Each one works on his site, user1 creates a CRCcard called "Model". This operation respects strictly the constraint defined by r1. The user user2 makes the same thing, he creates a CRC card, which is also named as "Model". This
operation also respects the constraints locally. However, after the integration of the distant operations of the user1 on the site of user2, the constraint is violated, since we have two cards with the same name (we have the same scenario on the site of the user2). This means that we must also check the constraints after the integration of the distant operations.

In summary, the constraints must be checked after the execution of a local macro operation and after integration of a macro distant operation.

Constraints Checking
To check the constraints, we use the algorithm implemented in xlinkit. When we check the consistency of the document "doc.xml" against constraint defined previously, we obtain the following file:

```xml
<xlinkit:ConsistencyLink
   ruleid="rule.xml#/consistencyruleset/consistencyrule[1]"
xlink:type="extended">
   <xlinkit:State>inconsistent</xlinkit:State>
   <xlinkit:Locator number="1"
xlink:href="doc.xml#/ROOT/card[1]"
xlink:type="locator"/>
   <xlinkit:Locator number="2"
xlink:href="doc.xml#/ROOT/card[2]"
xlink:type="locator"/>
</xlinkit:ConsistencyLink>
```

This means that there is an inconsistency between card1 and card2 of the document "doc.xml": these two cards have the same name.

Constraints Violation
Classical solution for this kind of problem is to abort the transaction that violates the constraints. In our context, aborting a macro operation is difficult because it can be a remote one. Aborting a macro operation can have effects not only on the current site but also in all other sites that received the operation. For this problem, we can use the Undo algorithm developed in [21].

Another possible solution could be to provide users with information about violated constraints and objects concerned with this violation and thus allowing them to do more operations to compensate this violation. In the future, we can provide users with procedures to repair the inconsistency.

If we apply these two solutions with our example of CRC cards, the first solution will undo one CRC card creation, the other will mark CRC cards as violating the constraints. Then, users can compensate by renaming or undoing one CRC card creation. The system can itself repair the conflicting state by automatically renaming one CRC cards. In our implementation, we have chosen the compensation approach.

IMPLEMENTATION
To validate our approach, we integrate the constraints in the environment SAMS. SAMS [4] allows a team to work on the same project, on distant geographical sites. It can be tested online: http://woinville.loria.fr/sams. Figure 6 shows the graphical interface of SAMS.

Figure 4: SAMS environment
SAMS is based on typed XML object. Two editors specialized on CRC card and documents HTML are implemented. This environment implements the primitives operations CreateNode, DeleteNode and their transformation functions. Figure 5 presents the class diagram of SAMS operations.

Figure 5: SAMS primitive commands
As we said before, we add new object Constraint with predefined primitive commands and their transformation functions. Figure 6 gives the new SAMS diagram.
any precision about rules, how to define them, about a grammar in text editor. However, authors do not give during the collaborative work, for example, these rules can this operation at all sites abide by rules defined before or consistency for any operation, the effect of execution of difference is the semantic consistency. Semantic consistency promises the rules defined before or promises the intentions of operations are kept, and execution order and result at all sites, content consistency and semantic consistency. Operation consistency promises the same operation consistency, content consistency and semantic consistency. The new consistency model has three parts: for describing the consistency issue in collaborative editing and deficiencies, which may not be the best previous consistency model based on causality, convergence and intention preservation. According to the authors the previous consistency model shows ambiguities and deficiencies, which may not be the best suitable one for describing the consistency issue in collaborative editing systems. The new consistency model has three parts: operation consistency, content consistency and semantic consistency. Operation consistency promises the same execution order and result at all sites, content consistency promises the intentions of operations are kept, and semantic consistency promises the rules defined before or during the collaborative work are abided. If we compare the new model with the old one, the only difference is the semantic consistency. Semantic consistency for any operation, the effect of execution of this operation at all sites abide by rules defined before or during the collaborative work, for example, these rules can be grammar in text editor. However, authors do not give any precision about rules, how to define them, about a mechanism to verify these rules as we do in this paper. Authors argue that their model is far from maturity. This is the first consistency mechanism proposed in the context of transformational approach.

In this paper, we do not want to replace the consistency model of transformational approach. We define semantic consistency model that can be integrated with the syntactic consistency model of the transformational approach. In real-time groupware systems, sun et al [5] points the importance of future research for exploring the issues and techniques related to semantic consistency in operational transformation approach.

CONCLUSION AND FUTURE WORKS
In this paper, we have shown from theoretical and practical point of view that it is possible to integrate constraints with an operational transformation approach. This allows ensuring general properties: causality, convergence and preserving user intention while ensuring semantic consistency. To validate our proposition, we have integrated the constraints in the SAMS environment. Adding constraints to operational transformation approach allows to ensure semantic consistency in addition to the syntactic consistency.

We have also pointed out the synergy that exists between the transformational approach and the semantic constraints. There is a tradeoff between defining high-level operations and required transformation functions and low-level operations with semantic constraints. The second solution is simpler because defining transformation functions is a complex task. This approach let us work with generic low-level operation while handling high level semantic.

We are working on several perspectives:

- Checking the consistency of constraints themselves. This requires to use theorem prover.
- Managing dynamic constraints.

ACKNOWLEDGMENTS
Our thanks to Lionel Verscheu, student of University Henri Poincaré, for his contribution in the implementation of the constraints in SAMS environment and to Sébastien Jourdain, expert engineer at ECOO team, and Co-developer of SAMS environment for his help. And many thanks for the anonymous reviewers of this paper.

REFERENCES
2. Maher Suleiman, Michèle Cart, and Jean Ferrié. Concurrent operations in a distributed and mobile


12. Wenfei Fan, Jérôme Siméon; Integrity Constraints for XML. *PODS 2000*.


22.