A Proposal for Transactions in the Semantic Web

Ana Sofia Gomes and José Júlio Alferes

Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Abstract. The success of the Semantic Web project has triggered the emergence of new challenges for the research community. Among them, relies the ability of evolving the web by means of actions and updates in accordance with some standard proposals as RIF or SPARQL-Update. However, from the moment that actions and updates are possible, the need to ensure properties regarding the outcome of performing such actions emerges. Moreover, this need also leaves open the specification of such properties and requirements that an intended solution should comply to.
In this paper we motivate the need for employing transactional properties in this new Web and delineate a proposal for the requirements that such solution should provide. Afterwards, we develop a logic, based on the well-known Transaction Logic, that partially achieves such requirements, as a first step of an ongoing work.

1 Introduction

The World Wide Web has dramatically changed the way we communicate and share knowledge by aiding all kinds of users to access documents and participate with their own content. This, along with the simplicity inherent of the Web (which is in fact the most crucial factor for its popularity), enabled the explosion of both range and quantity of Web contents.

However, this sheer growth has triggered the appearance of more efficient ways to locate and search resources, and as a result, in the past decade, a large trend has been developed to evolve from a static and informative web to a linked web of data. Traditionally, data published on the Web was designed mainly for humans and made available in strict formats such as CSV or XML, or marked up as HTML tables, sacrificing much of its structure and semantics. As a consequence, the exploration of web content by automated processes became a synonym of a complex and painful task.

A community effort started with the advent of the Semantic Web, aims to disrupt this tendency by explicitly defining the semantics of contents and links for machine consumption. Underpinning this evolution is a demand to provide data as "raw" and thereby enabling the construction of powerful linked data mashups

across heterogeneous data source collections, without further programming effort. This project known as Linked Data[1] envisages to link arbitrary things in the web, relying mainly on documents containing data in RDF [**?**] (Resource Description Framework) format. This movement has gained such popularity that today the amount of links between datasets as well as the quality of these links has largely increased, paving the way to a *Web of Data* with the ultimate goal to use the web like a single global database. Nevertheless, in order to achieve such realization there are still several research quests that need to be addressed, and before using the web as a huge database it is essential to ensure (at least some) properties that one is used to see in standard databases [**?**]. Among these features is the ability to perform transactions which is the subject of this paper.

2 Transactions on a Web of Data

2.1 The Evolving Web

The growth of popularity of the semantic web has triggered the appearance of query languages capable of extracting knowledge from a complete distributed and heterogeneous database. One example is the SPARQL language [**?**] endorsed by W3C and able to perform SQL-like queries in RDF. This kind of languages has the power to relate and extract information from completely different applications, sites and/or services such as RSS feeds, government data or individual Friend of a Friend files.

Nonetheless, the (semantic) web, as envisioned, should be able to perform more activities than just querying. Communication platforms such as wikis (where several users can modify the same document), or online market places, are examples of existing web applications that require updates according to client requests or actions. However, if today's web evolution means evolution of individual web sites that are updated locally, lots of effort has been made to embrace the idea of cooperative evolution [**?**].

Crucial to the success of the semantic web is a cooperative behavior to publish web content in standard format (OWL [**?**] and RDF) in order to make intelligent access possible. Likewise, cooperative evolution relates to a joint effort to provide the web with the tools it needs in order to automatically evolve. Consequently, along with OWL and RDF standards, others have been proposed to break up with the conventional idea that the web is for read-only operations. In fact, the same way it is intended to give intelligent agents tools to consult the web, it is also the goal to allow these intelligent agents to perform automatic updates according to some rules. One example of this is the RIF-PRD [4] specification that intends to provide semantics for executing actions on the web in accordance with some given production rules (**If** *Condition* **then** *Action*).

In a similar way, other proposals have been made to give the web not only the capability to act but also to react automatically to changes and events. In this context, Event-Condition-Action languages, which represent an intuitive

[1] `http://linkeddata.org/`

and powerful paradigm for programming reactive system, have been largely used to provide such semantics to the web [**?**,**?**,**?**].

As a result, from the original proposal of the Semantic Web by Tim Berners-Lee in 1998 [**?**], a big effort has been made from the research community to shift from a static web to a network of autonomous data sources capable of reacting to changes and self-updating [**?**,**?**,**?**,**?**].

2.2 Motivating Transactions on the Web

With the adoption of the web of data as a new paradigm for the web, several problems arise. Particularly, from the moment that actions and updates are possible, it appears the need to ensure some properties regarding the outcome of performing such actions. As an illustration, imagine some RIF-PRD production rule stating that: ***If*** *a customer reaches $5000 of cumulative purchases during the current year* ***then*** *its status becomes* Gold *and a golden customer card will be printed and sent to him within one week.* In RIF-PRD syntax this is translated into:

```
Prefix(ex <http://example.com/2008/prd1#>)
Forall ?customer ?purchasesYTD (
 If   And( ?customer#ex:Customer
           ?customer[ex:purchasesYTD->?purchasesYTD]
           External(pred:numeric-greater-than(?purchasesYTD 5000)))
 Then Do( And( Modify(?customer[ex:status->"Gold"])
               Execute(act:printCard(?customer,"Gold")))))
```

One obvious requirement of applying such rule is atomicity, that is, if the action could not be performed completely, then it should not be performed at all. In this particular example, a customer should not become a gold customer without the emission of the corresponding card, neither a card should be delivered to a customer whose status is not *gold*.

This kind of problems is generally solved in databases with the use of transactions. Transactions ensure atomicity, consistency, isolation and durability of a special set of actions. These properties, known as ACID, play a fundamental role in providing reliability to standard databases. Atomicity requires a transaction to follow an all-or-nothing rule – all operations should be performed as a unit which means that if one part of the transaction fails then the entire transaction must fail as well. The standard way to handle a failure of an ACID transaction is by a rollback, i.e. by restoring the state of the system before the execution of the failed transaction. Consistency ensures that a transaction either achieves a state where consistency of data is preserved, or returns to the original (consistent) state without changing data. Isolation guarantees that, even though a set of transactions can be executed concurrently, the outcome is equivalent to execute each transaction one-by-one in a given sequence. Durability states that

once a transaction has been committed, its changes will not be lost, even in the event of some system failure[2].

2.3 Why the ACID model is not enough

Albeit the advantages of the ACID model, it imposes severe demands [5] that are not always suitable for some systems, e.g. Web Sources, as argued in [**?**,**?**].

Particularly, atomicity requires all the steps of the transactions to rollback when the transaction fails. However, when these steps include external actions like sending an email or printing some document, this property is no longer possible to be guaranteed, since there is no way to revert these operations. In addition, there are situations where a transaction may involve iterated information exchange between different actors such as web services, human agents, databases, etc., potentially lasting for hours or days. These transactions are denoted as *long running transactions* or *sagas* [5]. Since it is impossible to perform a rollback when a transaction with such characteristics fails, other mechanisms are required. The usual approach is to define *compensation* operations for each operation to be performed. The idea is that these compensations lead the database into a state that is considered equivalent to the initial one, thus achieving some weaker form of atomicity. It is worth noting that the traditional rollback of an ACID transaction can be seen also as a form of compensation.

Since the traditional ACID model as found in standard databases is not suitable for the new context of the Semantic Web, we argue that transaction's properties must be redefined taken into consideration the characteristics of this peculiar context. Nevertheless, it is unquestionable that ensuring at least a relaxed model of such properties will provide crucial reliability to this emergent web of data.

2.4 Requirements for a new Web

After motivating the need for implementing transactional properties on the web, and discussing the limitations of the standard ACID model, we now present some argument on the necessary requirements for a new transactional model aimed for the new Web.

Transparency and collaboration are key features of anything that is related to the Web. One example of this concern is the W3C Recommendation RIF [**?**] which was designed to provide a standard for rule interchange in the web. In such context, declarative languages can play a fundamental role by providing ways to define programs which are clear and quickly understandable by their users. In fact, declarative languages have the advantage of being substantially more concise and self-explanatory, as they state what is to be computed rather than how it is to be computed. As a result, they are inherently high-level where

[2] Durability is usually achieved by low-level software management and is hence outside the scope of this study.

programs can be viewed as theories and the details of the computation are left to the abstract machine.

However, as motivated by the proposal of several dialects for RIF, a language for the web needs not only to provide means to represent and reason about knowledge, but also to allow the execution of rules and actions. In this sense we are aiming for a declarative language that allow us to represent programs, but also to execute them.

Moreover, in order to abandon the concept of a read-only web, it is necessary to provide it the ability to react and respond to changes. In fact, the web is mainly dynamic, in the sense that its resources may change their content over time, and thus, a strong motivation exists in defining languages to specify updates and to detect them immediately. Furthermore, it is important to not only enable the detection of atomic events, but also of complex ones, in order for the solution to be useful in real scenarios.

Additionally, when defining a language which is intended to specify transactions it is obvious that this language must provide the ability to guarantee ACID properties, and to this end, it is necessary to have some notion of state change embedded in the theory. However, the web as a huge database consists of an agglomerate of different sources accessed and updated by an unpredictable large number of users, and thus it is impossible to control all the knowledge as well as to guarantee its consistency. As a result, if by one hand it is important to ensure ACID properties for local usage where one has total (or at least a high) control of the data and its accesses; on the other hand, and continuing the argumentation provided in Section 2.3, we believe that these properties are too strong in cases where it is not intended to restrict who can update the knowledge base, and/or what is updated. Note that this coincides with the reasons for combining open-world assumption with closed-world assumption for the semantic web context [**?**]. This way, our argumentation is that the two possibilities must coexist together, and the intended semantics must allow one to switch between pure database transactions (which ensure all ACID properties), and a weaker model of transaction designed for the web and loosening some properties like Isolation and Atomicity.

Finally, it is necessary to take into account that the Web is accessed by an unpredictable large number of users. This way, anything conceived for this kind of context, needs to be scalable and provide concurrency features. Nevertheless, it is worth noting that we are not interested here on "low-level" algorithms or efficient implementations for the problem of integrating transactions in the Semantic Web context. Particularly, the main goal of this proposal is concerned with knowledge representation and what are the requirements that a language and semantics must have in order to express transactions for this context. Therefore, when we refer to properties such as scalability and concurrency, we are arguing that these properties must be part of the intended semantics in a similar sense as Concurrent Transaction Logic ($\mathcal{CTR}$) [2] or the Calculus of Communicating Systems (CCS) [**?**].

In summary, we believe that a semantics which defines transactional properties needs to provide the following properties in order to be considered suitable for the context of a Web of Data: (1) *Declarativity*; (2) *Reactivity*; (3) *Transactional Properties* (ACID model); (4) *Weaker model of Transactional Properties*; (5) *Concurrency*.

3 A logic for transactions with external actions

As a first step to achieve the requirements proposed we propose a novel logic that allows for the combination of standard ACID transactions with external actions. Such logic has two main components, an "internal" component following the standard ACID model that interacts and executes actions with an "external" component. However, since it is impossible to rollback operations in a system that is *external*, the logic ensures a relaxed form of atomicity in the external domain by means of compensation operations.

As a starting point of the logic, we use Transaction Logic, a unique logic for specifying transactions in a very flexible way.

3.1 Transaction Logic

Transaction Logic ($\mathcal{TR}$) is an extension of predicate logic originally proposed in [1] which provides a logical foundation to reason about state changes in arbitrary logical theories (such as databases, logic programs or other knowledge bases), and particularly, to deal with ACID transactions. Contrary to most logics that reason about state change, $\mathcal{TR}$ does not use a separate procedural language to specify programs, as programs are specified in the logic itself. $\mathcal{TR}$ is thus a single representation language that can be used in two ways: to reason about programs (and the properties that they need to satisfy), and to execute them. When used for reasoning, one can, for instance, infer that a particular program preserves the integrity constraints of a knowledge base; or that under certain conditions, a transaction program is guaranteed not to abort. $\mathcal{TR}$ thus comes with a natural model theory (to perform reasoning) and a sound and complete proof theory (to specify and execute programs).

Moreover, reasoning in $\mathcal{TR}$ is *flexible* in the sense that it does not commit to any particular logical theory. To achieve this flexibility, $\mathcal{TR}$ is parameterized by a pair of oracles that encapsulate elementary knowledge base operations of querying and updating, thus allowing $\mathcal{TR}$ to reason about states and updates while accommodating a wide variety of semantics [1]. As a result, there is no distinction in $\mathcal{TR}$ between formulas that query the knowledge base and formulas that update it. As in classical logic, every formula has a truth value, but it also may have a side effect by changing the state of the knowledge base. It is thus the oracles' responsibility to decide if the formula can be executed and its corresponding effects.

Example 1 (Financial Transactions). As illustration of $\mathcal{TR}$, consider a knowledge base of a bank [1] where the balance of an account is given by the relation

$balance(Acnt, Amt)$. To modify it we have a pair of elementary update operations: $balance(Acnt, Amt).ins$ and $balance(Acnt, Amt).del$ (denoting the insertion, resp. deletion, of a tuple of the relation). With these elementary updates, one may define several transactions, e.g. for making deposits in an account, make transfers from one account to another, etc. In $\mathcal{TR}$ one may define such transactions by the rules below where, e.g the first one means that one possible way to succeed the transfer of Amt from $Acnt$ to $Acnt'$ is by first withdrawing Amt from $Acnt$, followed by (denoted by $\otimes$) depositing Amt in $Acnt'$.

$$transfer(Amt, Acnt, Acnt') \leftarrow withdraw(Amt, Acnt) \otimes deposit(Amt, Acnt')$$
$$withdraw(Amt, Acnt) \leftarrow balance(Acnt, B) \otimes changeBalance(Acnt, B, B - Amt)$$
$$deposit(Amt, Acnt) \leftarrow balance(Acnt, B) \otimes changeBalance(Acnt, B, B + Amt)$$
$$changeBalance(Acnt, B, B') \leftarrow balance(Acnt, B).del \otimes balance(Acnt, B').ins$$

State change and evolution in $\mathcal{TR}$ is caused by executing ACID transactions, i.e. by posing logical formulas into the system in a Prolog-like style as e.g. $? - transfer(10, a_1, a_2)$. Since every formula is assumed as a transaction, by posing $transfer(10, a_1, a_2)$ we know that either $transfer(10, a_1, a_2)$ can be executed respecting all ACID properties evolving the knowledge base from an initial state D_0 into a state D_n (passing through an arbitrary number of states n); or $transfer(10, a_1, a_2)$ cannot be executed under these conditions and so the knowledge base does not evolve and remains in the state D_0. Also, in $\mathcal{TR}$ it is possible to have several rules (or rule instances) for defining one transaction, thereby allowing for the specification non-deterministic transactions.

3.2 External Transaction Logic

The characteristics shown make $\mathcal{TR}$ a unique logic for specifying transactions as it provides a unifying framework combining declarative knowledge *and* execution, whilst achieving a high flexible semantics. As argued, these are some of the features desired for transactional languages for the web.

Unfortunately $\mathcal{TR}$ is not suitable to model situations that require relaxing the standard ACID model. This limitation makes it impossible to express in $\mathcal{TR}$ *external actions*, that is, actions that are executed in an external entity. Furthermore, since $\mathcal{TR}$ only considers internal knowledge, i.e. transactions can only be executed in an internal knowledge base where one has a complete control and there is no way to interact with external entities, it becomes impossible for $\mathcal{TR}$ to *react* to external changes. In fact, as it is, $\mathcal{TR}$ is passive in the sense that transactions are only executed upon requests and internally. This conventional pattern has already been considered insufficient by the database community [**?**], and today most DBMS provide reactive features.

With the goal to provide a solution for the aforementioned limitations, and as a first step in achieving the requirements proposed in Section 2.4 we propose External Transaction Logic ($\mathcal{ETR}$). $\mathcal{ETR}$ is an extension of $\mathcal{TR}$ to accommodate interactions with an external domain. This interaction requires relaxing the traditional ACID model, since one has no control over the *external* domain in which actions are executed, external actions cannot be rollbacked, and thus, it is

no longer possible to ensure the standard ACID model. To address this, $\mathcal{ETR}$ follows the proposal of [5]. The idea is to define a compensation for each operation to be performed. If the transaction fails and these compensations are performed in backward order, then they lead the database into a state that is considered equivalent to the initial one, thus ensuring a weaker form of atomicity.

3.3 Syntax and Oracles

$\mathcal{ETR}$ operates over a knowledge base which includes both an internal knowledge base, and an external domain, on which actions may be performed. For that, formally $\mathcal{ETR}$ works over two propositional languages: $\mathcal{L}_P$ (states language), and $\mathcal{L}_a$ (action language). Propositions in $\mathcal{L}_a$ denote actions that can be executed in the external domain. Propositions in $\mathcal{L}_P$ represent fluents that are true (or false) in the internal knowledge base, as well as in the external domain.

To build complex logical formulas, $\mathcal{ETR}$ uses the usual classical logic connectives (of conjunction, disjunction, etc) plus a special connective $\otimes$ to denoted *serial conjunction*. Informally, the formula $\phi \otimes \psi$ represents an action composed of an execution of ϕ followed by an execution of ψ. To allow for the specification of external actions $\mathcal{ETR}$ uses a special kind of formula $\mathbf{ext}(a, a^{-1})$ known as *external*. In $\mathbf{ext}(a, a^{-1})$, a is an external action formula, and a^{-1} its corresponding compensation which can be internal and/or external actions as to make possible for more flexible compensations.

Example 2. Consider the system of the web shop where clients submit orders. In the end of each order, a final confirmation is asked to the client that may or not confirm the transaction. If the client accepts it, the order ends successfully. Otherwise, the transaction fails and consistency must be preserved. In this case, it means that we need to rollback the update of the stock, and to compensate for the executed payment. The obvious compensation here is to simply ask the bank to refund the charged money. However, note that the transaction may fail sooner. E.g. the transaction may fail if the bank cannot charge the given amount in the credit card, or if the product is out-of-stock. This situation can be modeled in $\mathcal{ETR}$ by the rules:

$$buy(Prdt, Card, Amt) \leftarrow \mathbf{ext}(chargeCard(Card, Amt), refundCard(Card, Amt)) \\ \otimes\ updateStock(Prdt) \otimes\ \mathbf{ext}(confirmTransaction(Product, Card, Amt), ())$$

$$updateStock(Prdt) \quad \leftarrow product(Prdt, N, WHouse) \otimes\ N > 0 \\ \otimes\ product(Prdt, N, WHouse).del \otimes\ product(Prdt, N-1, WHouse).ins$$

To reason about elementary updates, $\mathcal{ETR}$ is parameterized by a triple of oracles $\mathcal{O}^d$, $\mathcal{O}^t$ and $\mathcal{O}^e$ respectively denoted the data, the transition oracle and the external oracle. These oracles encapsulate the elementary knowledge base operations, allowing the separation of elementary operations from the logic of combining them. As a result of this separation, $\mathcal{ETR}$ does not commit to any particular theory of elementary updates. An $\mathcal{ETR}$ program is then defined as follows.

Definition 1 ($\mathcal{ETR}$ actions, atoms, formulas and programs). *Given propositional languages $\mathcal{L}_P$ and $\mathcal{L}_a$, an $\mathcal{ETR}$* action *is either a proposition in $\mathcal{L}_a$, or* $\mathbf{ext}(a, b)$ *where a is a proposition in $\mathcal{L}_a$ and b is either a proposition in $\mathcal{L}_a$ or a proposition in $\mathcal{L}_P$. An $\mathcal{ETR}$* atom *is either a proposition in $\mathcal{L}_P$ or an $\mathcal{ETR}$ action. $\mathcal{ETR}$* formulas *are inductively defined as follows:*

- *an $\mathcal{ETR}$ atom is an $\mathcal{ETR}$ formula;*
- *if ϕ and ψ are $\mathcal{ETR}$ formulas, then $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \leftarrow \psi$ and $\phi \otimes \psi$ are $\mathcal{ETR}$ formulas; nothing else is an $\mathcal{ETR}$ formula.*

A set of of $\mathcal{ETR}$ formulas is called an $\mathcal{ETR}$ program.

3.4 Model Theory

An important concept in $\mathcal{ETR}$'s model theory is the notion of *compensation*. A compensation occurs when the executed transaction ϕ contains external actions and fails. Since, in such a case, it is not possible to simply rollback to the initial state before executing ϕ, a series of compensating actions are executed to restore the consistency of the external knowledge base.

A transaction is, as usual, a sequence of actions that need to be performed (among other things) in a all-or-nothing way, making the internal knowledge base evolve from an initial state D_1 into a state D_n. In $\mathcal{ETR}$, since an external domain is also considered, a transaction may also make this external domain evolve from an initial state E_1 into a state E_m. During the execution of such a transaction both the internal and the external domain pass through an arbitrary number of intermediate states $D_1, D_2, \ldots, D_{n-1}, D_n$ and $E_1, \ldots, E_m$. This notion of sequence of states, denoted as *path*, is central to $\mathcal{ETR}$'s model theory as it represents the basic structure on which formulas are evaluated.

Given an $\mathcal{ETR}$ theory, formulas are evaluated on paths of internal states (as also in the original $\mathcal{TR}$), together with paths of external states, and with sequences of (external) actions (as we shall see, needed to perform compensations). For that, an *interpretation* is defined as a mapping from such a pair of paths and a sequence of actions into a set of $\mathcal{ETR}$ formulas (those true under that interpretation). As in $\mathcal{TR}$, interpretations are restricted such that formulas classically true in a state are true in (internal) paths just with that state (evaluated by the state data oracle $\mathcal{O}^d$), and such that, if a formula φ forces the internal state to evolve from D_1 into D_2 (evaluated by the state transition oracle $\mathcal{O}^t$), then it is true in the path D_1, D_2. Moreover, to account for the behavior of the external domain, we further restrict interpretations to obey to an external oracle $\mathcal{O}^e$ that detects external changes, and in which the external actions are modeled.

Definition 2 (Interpretations). *An interpretation is a mapping M that given a path of internal states, a path of external states and a sequence of actions, returns a set of transaction formulas (or $\top$)[3]. This mapping is subject to the following restrictions:*

[3] Similar to $\mathcal{TR}$, for not having to consider partial mappings, besides formulas, interpretation can also return the special symbol $\top$. The interested reader is referred to [3] for details.

1. $\varphi \in M(\langle D\rangle, \langle E\rangle, \emptyset)$, *for every* φ *such that* $\mathcal{O}^d(D) \models \varphi$
2. $\varphi \in M(\langle D_1, D_2\rangle, \langle E\rangle, \emptyset)$ *if* $\mathcal{O}^t(D_1, D_2) \models \varphi$
3. $A \in M(\langle D\rangle, \langle E_1, \ldots, E_p\rangle, \langle A\rangle)$
   if $\mathcal{O}^e(E_1, \ldots, E_p) \models A \quad$ *and* $\quad p > 1$

The definition of satisfaction of $\mathcal{ETR}$ formulas, over general paths, requires the prior definition of operations on paths. These take into account how sequences of action are satisfied, and how to construct the correct compensation.

Definition 3 (Paths and Splits). *A* path *of length* k*, or a* k-path*, is any finite sequence of states (where the* S*s are all either internal or external states),* $\pi = \langle S_1, \ldots, S_k\rangle$*, where* $k \geq 1$*. A* split *of* π *is any pair of subpaths,* π_1 *and* π_2*, such that* $\pi_1 = \langle S_1, \ldots, S_i\rangle$ *and* $\pi_2 = \langle S_i, \ldots, S_k\rangle$ *for some* i $(1 \leq i \leq k)$*. In this case, we write* $\pi = \pi_1 \circ \pi_2$*.*

Definition 4 (External action split). *A split of a sequence of external actions* $\alpha = \langle A_1, \ldots, A_j\rangle$ $(j \geq 0)$ *is any pair of subsequences,* α_1 *and* α_2*, such that* $\alpha_1 = \langle A_1, \ldots, A_i\rangle$ *and* $\alpha_2 = \langle A_{i+1}, \ldots, A_j\rangle$ *for some* i $(0 \leq i \leq k)$*. In this case, we write* $\alpha = \alpha_1\alpha_2$*.*

Note that there is a significant difference between Definitions 3 and 4. In fact, splits for sequences of external actions can be empty, and thus, it is possible to define splits of empty sequences, whereas a split of a path requires a sequence with at least length 1.

Besides the general definition of paths and splits of states and actions, we also define special operations over internal paths and external actions to handle compensations. The idea is that, if a transaction formula that contains external actions fails, then the compensations of each external action performed need to be executed in the backward order and the internal path rollbacked, i.e. the initial internal state is restored as the current state. These notions are made precise as follows.

Definition 5 (Rollback split). *A rollback split of* $\pi = \langle D_1, \ldots, D_k\rangle$ *is any pair of finite subpaths,* π_1 *and* π_2*, such that* $\pi_1 = \langle D_1, \ldots, D_i, D_1\rangle$ *and* $\pi_2 = \langle D_1, D_{i+1}, \ldots, D_k\rangle$*.*

Definition 6 (Inversion). *An external action inversion of a sequence* α *where* $\alpha = (\mathbf{ext}(a_1, a_1^{-1}), \ldots, \mathbf{ext}(a_n, a_n^{-1}))$*, denoted* α^{-1}*, is the corresponding sequence of compensating external actions performed in the inverse way as* $(a_n^{-1}, \ldots, a_1^{-1})$*.*

Note that inversion is only defined for sequences where all action have are of the form $\mathbf{ext}(a, a^{-1})$. In fact, if for one action in the sequence no compensation is defined, then it is impossible to compensate the whole sequence. Building on these definitions, we formalize what (complex) formulas are true on what paths.

Definition 7 (Satisfaction). *Let* M *be an interpretation,* π *be an internal path,* ϵ *be an external path and* α *be a sequence of external actions. If* $M(\pi, \epsilon, \alpha) = \top$ *then* $M, \pi, \epsilon, \alpha \models \phi$ *for every* $\mathcal{ETR}$ *formula* ϕ*; otherwise:*

1. ***Base Case:*** $M, \pi, \epsilon, \alpha \models p$ *if* $p \in M(\pi, \epsilon, \alpha)$ *for any* $\mathcal{ETR}$ *atom* p
2. ***Negation:*** $M, \pi, \epsilon, \alpha \models \neg\phi$ *if it is not the case that* $M, \pi, \epsilon, \alpha \models \phi$
3. ***"Classical" Conjunction:*** $M, \pi, \epsilon, \alpha \models \phi \wedge \psi$ *if* $M, \pi, \epsilon, \alpha \models \phi$ *and* $M, \pi, \epsilon, \alpha \models \psi$.
4. ***Serial Conjunction:*** $M, \pi, \epsilon, \alpha \models \phi \otimes \psi$ *if* $M, \pi_1, \epsilon_1, \alpha_1 \models \phi$ *and* $M, \pi_2, \epsilon_2, \alpha_2 \models \psi$ *for some split* $\pi_1 \circ \pi_2$ *of path* π*, some split* $\epsilon_1 \circ \epsilon_2$ *of path* ϵ*, and some external action split* $\alpha_1 \circ \alpha_2$ *of external actions* α.
5. ***Compensating Case:*** $M, \pi, \epsilon, \alpha \models \phi$ *if* $M, \pi_1, \epsilon_1, \alpha_1\alpha_1^{-1} \rightsquigarrow \phi$ *and* $M, \pi_2, \epsilon_2, \alpha_2 \models \phi$ *for some split* $\pi_1 \circ \pi_2$ *of* π*, some split* $\epsilon_1 \circ \epsilon_2$ *of path* ϵ*, and some external action split* $\alpha_1\alpha_1^{-1}$*,* α_2 *of* α.
6. *For no other* $M, \pi, \epsilon, \alpha, \phi$ *it holds that* $M, \pi, \epsilon, \alpha \models \phi$.

In the sequel we also mention the satisfaction of disjunctions and implications, where as usual $\phi \vee \psi$ means $\neg(\neg\phi \wedge \neg\psi)$, and $\phi \leftarrow \psi$ means $\phi \vee \neg\psi$.

Note that Definition 7 requires the definition of *Consistency Preserving Path*. Intuitively, $M, \pi, \epsilon, \alpha\alpha^{-1} \rightsquigarrow \phi$ (defined below) means that, in the *failed* attempt to execute ϕ, a sequence α of external actions were performed. Although the internal state before the execution of the transaction is restored (by rollback split), since it is impossible to perform external rollbacks, consistency is ensured by performing a sequence of compensating actions α^{-1} in backward order (in case such compensating actions were defined, i.e. if there is an inversion of α). A formula ϕ is said to succeed over a compensating case if, although the execution failed, it is possible to construct a consistency preserving path (cf. Definition 8) and further *succeed* on an alternative execution.

Definition 8 (Consistency Preserving Path). *Let* M *be an interpretation,* π *be an internal path,* ϵ *an external path, and* α *be a* non-empty *sequence of external actions such that* α^{-1} *is defined. Let* π_1 *and* π_2 *be a rollback split of* π*. The path* π_1' *is obtained from* $\pi_1 = \langle D_1, \ldots, D_n\rangle$ *by removing the state* D_n *from the sequence;* α^{-1} *is a non-empty sequence of external actions obtained from* α *by inversion;* $\epsilon_1 \circ \epsilon_2$ *is a split of* ϵ*. We say that* $M, \pi, \epsilon, \alpha\alpha^{-1} \rightsquigarrow \phi$ *iff* $\exists b_1 \otimes \ldots \otimes b_i \otimes \ldots \otimes b_n$ *such that:*
$M, \pi_1', \epsilon_1, \alpha \models \phi \leftarrow (b_1 \otimes \ldots \otimes b_i \otimes \ldots \otimes b_n)$
$M, \pi_1', \epsilon_1, \alpha \models b_1 \otimes \ldots \otimes b_i \otimes \neg\, b_{i+1}$
$M, \pi_2, \epsilon_2, \alpha^{-1} \models \bigotimes \alpha^{-1}$
where $\bigotimes$ *represents the operation of combining a sequence of actions using* $\otimes$.

The notion of satisfaction allows us to define models of $\mathcal{ETR}$ programs, and entailment of $\mathcal{ETR}$ formulas, in a natural way. Intuitively, a formula ϕ entails another formula ψ if, independently of the sequence of actions and internal and external paths, whenever ϕ is true ψ is also true; similarly for entailment of formulas by programs (i.e. sets of formulas). Formally:

Definition 9 (Models). *An interpretation* M *is a model of a transaction formula* ϕ *if* $M, \pi, \epsilon, \alpha \models \phi$ *for every internal path* π*, every external path* ϵ*, and every action sequence* α*. In this case, we write* $M \models \phi$*. An interpretation is a model of a set of formulas if it is a model of every formula in the set.*

Definition 10 (Logical Entailment). *Let P be an $\mathcal{ETR}$ program and ϕ be an $\mathcal{ETR}$ formula. Then P entails ϕ if every model of P is also a model of ϕ. In this case we write $P \models \phi$*

4 Comparisons and Related Work

$\mathcal{ETR}$ can be compared to many logics that reason about state change or about the related phenomena of time and action. These include action languages, the situation calculus [9], the event calculus [8], process logic [6] and many others. An extensive comparison of these formalisms with $\mathcal{TR}$ can be found in [1]. However, this kind of logics was not designed to reason about database programs but rather intended to describe changes in dynamic systems where one has little or no control such as external domains. As a result, although these formalisms provide powerful tools to specify changes and reason about their causalities in a very general and abstract way, they are simply inappropriate to model database transactions [1]. Moreover, the flexibility achieved by having an external oracle as a parameter allows for the combination of $\mathcal{ETR}$ with several different languages and semantics for describing the effects of actions in an external knowledge base. As a result, it is our opinion that rather than an alternative to $\mathcal{ETR}$, these solutions, as action languages or situation calculus, should be seen as a possible built-in component, orthogonal to $\mathcal{ETR}$ theory, as they can be used to define the semantics of the external oracle.

On the other hand, one can also compare $\mathcal{ETR}$ to formalisms that involve the notion of long-running transactions or sagas. Generally such formalisms are based on process algebras, a family of algebraic systems for modeling concurrent communicating processes, as Milner's Calculus of Communicating Systems (CCS) and Hoare's Communicating Sequential Processes (CSP), among others. One clear difference between $\mathcal{ETR}$ and such systems is that $\mathcal{ETR}$ does not support concurrency and synchronization. However, extending $\mathcal{ETR}$ to provide such features represents a next obvious step and is in line with what has been done in Concurrent Transaction Logic [2].

Notwithstanding the major difference between $\mathcal{ETR}$ and other proposals based on process algebras as [7,10] is mainly conceptual. In fact, the semantics of these latter systems are mostly focused on the correct execution and synchronization of processes whilst $\mathcal{ETR}$ semantics emphasizes knowledge base states. As a result, process algebras solutions are interested in modeling the correctness evolution of each transaction, thereby possessing a powerful operational semantics, but they are normally not interested in knowing what is true in each state of the knowledge base. In this sense, such solutions enclose powerful operators that in some cases even allow the system to construct the correct compensation for each action "on-the-fly" as in [11]. However, since these solutions, based on process algebras, are designed to define programs and behaviors, they are not suitable to be used as a knowledge representation formalism. Consequently, it is not possible to model what is true at each step of the execution of these processes nor to specify constraints on their execution based on this knowledge.

5 Discussion and Future Work

This work represents a first step towards a more generic goal, that we intend to pursue in our future work. Particularly we are interesting in developing a generic solution able to ensure the requirements as presented in Section 2.4. With this goal we already started to define top-down procedures for the serial-Horn subset of the logic.

Moreover, as it is, $\mathcal{ETR}$ does not support concurrency and synchronization. However, extending $\mathcal{ETR}$ to provide such features represents a next obvious step and is in line with what has been done in Concurrent Transaction Logic [2].

Another important step is to address reactivity. Reactivity denotes the ability to monitor changes and act accordingly to them. Such issue has always been an important concern and several active database language and systems have been proposed so far - a very incomplete list include [**?**,**?**,**?**,**?**,**?**,**?**]. Normally, reactive system are based on an Event-Condition-Action (ECA) paradigm. ECA-rules have the general syntax: **on** *event* **if** *condition* **do** *action*. Intuitively, events are received as an input stream from the external environment. It is then the system's responsibility to interpret which events have occurred and identify which rules should be triggered. The condition is a query to check if the system is in a specific state, where the rule can be applied. Finally, the action part specifies the actions that should be performed after the event occurs and the condition is true. $\mathcal{TR}$ as defined is not suitable as the action component of such reactive language as it does not account the possibility to interact with the external world, and thus it becomes impossible to "receive" events as well as execute actions externally. Contrarily, by having the possibility to interact with an external oracle, $\mathcal{ETR}$ is able to perceive the effects of the external actions performed, but also to model other arbitrary changes that have independently occurred in that domain. These characteristics make $\mathcal{ETR}$ an ideal candidate for modeling the semantics of the action component of an ECA language, providing the possibility of combining internal ACID transactions with a relaxed model of transactions for the accommodation of external actions.

Acknowledgements

This paper was submitted to SDIA 2011 - 3rd Doctoral Symposium on Artificial Intelligence and is part of an ongoing PhD supervised by José Júlio Alferes started on February 2010 and foreseen to be concluded on February 2014. Ana Sofia Gomes is supported by the FCT grant SFRH / BD / 64038 / 2009.

References

1. A. J. Bonner and M. Kifer. Transaction logic programming (or a logic of declarative and procedural knowledge). Technical Report CSRI-323, Computer Systems Research Institute, University of Toronto, 1995.

2. A. J. Bonner and M. Kifer. Concurrency and communication in transaction logic. In *JICSLP*, pages 142–156, 1996.
3. A. J. Bonner and M. Kifer. Results on reasoning about updates in transaction logic. In *Transactions and Change in Logic Databases*, pages 166–196, 1998.
4. C. de Sainte Marie, G. Hallmark, and A. Paschke. RIF Production Rule Dialect, June 2010. W3C Recommendation `http://www.w3.org/TR/rif-prd/`.
5. H. Garcia-Molina and K. Salem. Sagas. *SIGMOD Rec.*, 16:249–259, December 1987.
6. D. Harel, D. Kozen, and R. Parikh. Process logic: Expressiveness, decidability, completeness. In *FOCS*, pages 129–142, 1980.
7. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
8. R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Comp.*, 4(1):67–95, 1986.
9. J. McCarthy. Situations, actions, and causal laws. Technical report, Stanford University, 1963. Reprinted in MIT Press, Cambridge, Mass., 1968 pages 410-417.
10. R. Milner. Calculi for synchrony and asynchrony. *Theor. Comput. Sci.*, 25:267–310, 1983.
11. C. Vaz and C. Ferreira. Towards compensation correctness in interactive systems. In *WS-FM*, pages 161–177, 2009.