

XA2C: a framework for manipulating XML data

Gilbert Tekli, Richard Chbeir, Jacques Fayolle

▶ **To cite this version:**

Gilbert Tekli, Richard Chbeir, Jacques Fayolle. XA2C: a framework for manipulating XML data. International Journal of Web Information Systems (IJWIS), 2011, 7 (3), pp.240-269. 10.1108/17440081111165884 . hal-00650607

HAL Id: hal-00650607
https://hal.science/hal-00650607

Submitted on 11 Dec 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

XA2C-A FRAMEWORK FOR MANIPULATING XML DATA

GILBERT TEKLI

Telecom St Etienne, University Jean Monnet, 25 rue Dr Remy Annino, 42000, St Etienne, France
gilbert.tekli@univ-et-etienne.fr

RICHARD CHBEIR

Laboratoire LE2I (UMR - CNRS), Bourgogne University, 21000 Dijon, France
richard.chbeir@u-bourgogne.fr

JACQUES FAYOLLE

Telecom St Etienne, University Jean Monnet, 25 rue Dr Remy Annino, 42000, St Etienne, France
Jacques.fayolle@univ-et-etienne.fr

Purpose - XML has spread beyond the computer science fields and reached other areas such as, e-commerce, identification, information storage, instant messaging and others. Data communicated over these domains is now mainly based on XML. Thus, allowing non-expert programmers to manipulate and control their XML data is essential.
Methodology/approach - In the literature, this issue has been dealt with from 2 perspectives: (i) XML alteration/adaptation techniques requiring a certain level of expertise to be implemented and are not unified yet, and (ii) Mashups, which are not formally defined yet and are not specific to XML data, and XML-oriented visual languages are based on structural transformations and data extraction mainly and do not allow XML textual data manipulations. In this paper, we discuss existing approaches and present our XA2C framework intended for both non-expert and expert programmers and providing them with means to write/draw their XML data manipulation operations.
Findings - The framework is defined based on the dataflow paradigm (visual diagram compositions) while taking advantage of both Mashups and XML-oriented visual languages by defining a well founded modular architecture and an XML-oriented visual functional composition language based on colored petri nets allowing functional compositions. The framework takes advantage of existing XML alteration/adaptation techniques by defining them as XML-oriented manipulation functions. A prototype called XA2C is developed and presented here for testing and validating our approach.
Value - This paper presents a detailed description of an XML-oriented manipulation framework implementing the XCDL language.

***Keywords*:** Visual languages, Colored Petri Nets, Composition, XML data manipulation, Concurrency.

***Paper type*:** Research paper.

Introduction

The widespread of XML today has invaded the world of computers and is present now in most of its fields (i.e., internet, networks, information systems, software and operating systems). Furthermore XML has reached beyond the computer domain and is being used to communicate crucial data in different areas such as e-commerce, data communication, identification, information storage, instant messaging and others. Therefore, due to the extensive use of textual information transmitted in form of XML structured data, it is becoming essential to allow all kind of users to manipulate corresponding XML data based on specific user requirements. As an example, consider a journalist who works in a news company covering global events. The journalist wishes to acquire all information being transmitted by different media sources (television channels, radio channels, journals …) in the form of RSS feeds, filter out their content, based on the topic (s)he is interested in, and then compare the resulted feeds. Based on the comparison results, a report covering relevant facts of the event needs to be generated.

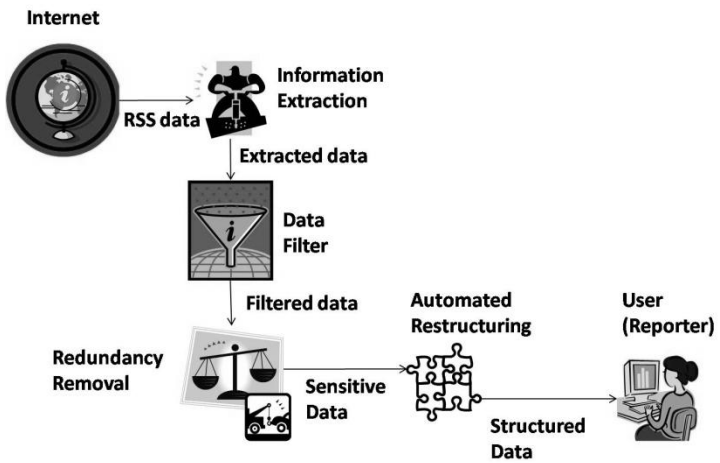


Fig.1: XML data manipulation scenario

In this first simple scenario, as shown in Figure 1, several separate techniques are needed to generate the manipulation operation required by the user such as XML filtering, string similarity comparison and automated XML generation. In a second scenario, consider a cardiologist who shares medical records of his patients with some of his colleagues and wishes to omit personal information concerning his patients (i.e., name, social security number, address, etc.). In this case, data omission is the manipulation required which can be done via data encryption, removal, substitution or others depending on the operations provided by the system and the requirements of the user (cardiologist in this case).

Based on these scenarios: (i) we need a framework for writing XML-oriented manipulation operations. It should contain all of the XML-oriented manipulation techniques. To the best of our knowledge, such a framework does not exist so far, and (ii) we need the framework to be used by both non-expert and expert programmers.

In order to address these 2 issues, 3 main approaches have emerged in the literature, XML Alteration/Adaptation techniques, Mashups and XML-oriented visual languages.

On one hand, while various Alteration/Adaptation techniques have emerged such as XML filtering (Altinel and Franklin, 2000), Adaptation (Pellan and Concolato, 2008) and Information Extraction (Chang and Lui, 2001), however we observed that these techniques share common functions but are defined each separately. They attempted to address specific requirements scoping different objectives. Whereas XML filtering is applicable to all XML data types and aims at filtering the data without any

alteration to the content, XML adaptation alters the data to adapt it to certain requirements but does not necessarily address all types of XML data. So far, each of these techniques remains separate from the other and no unified frameworks have been reached, not to mention that they require a high level of expertise for their implementation.

On the other hand, Both Mashups (Lorenzo et al., 2009) and XML visual languages (Braga et al., 2005) try to provide expert and non-expert users with the ability to write/draw data manipulations by means of visual elements. While there has been no common definition for Mashups, existing Mashup tools mainly aim at composing manipulation operators (e.g., RSS filters) for different types of web data (e.g., html, web site content…), but are not specific to XML. Since Mashups have not been formally defined, no languages have emerged yet providing visual functional compositions. On the other hand, XML oriented visual languages are already formalized and mainly based on existing XML transformation (e.g., XSLT) or querying languages (e.g., XQuery). They provide visual means for non expert programmers to write manipulation operations specific for XML data. Nonetheless, the expressiveness of existing XML-oriented visual languages is limited to their inability to visually express all the operations existing in the languages (e.g., aggregation functions) which they are based upon. Also the expressiveness is limited to the operations of these languages themselves. Their main goal is data extraction and structure transformation. Aside from their expressiveness limitations, on one hand, these languages normally require the user to have some knowledge in different

areas such as data querying which renders the task more difficult. On the other hand, they are not considered as visual functional composition languages.

Our research mainly aims at defining an XML-oriented framework allowing non-expert and expert users to write/draw and enforce XML manipulation operations based on functional composition. The functions can express but are not limited to alteration/adaptation techniques and are provided in forms of client libraries (e.g., DLL files) or online services (e.g., web-services). The framework is based on a visual functional composition language (Golin and Reiss, 1990), called XCDL (XML-Oriented Composition Definition Language). The language is based on the Dataflow paradigm and its syntax and semantics are defined based on Colored Petri Nets (CP-Nets) (Murata, 1989, Jensen, 1994) which allow it to express complex compositions with true concurrency (serial and parallel executions). In this paper, we introduce our XML alteration/adaptation control framework (Tekli et al., 2010a). We briefly present our composition language, XCDL (Tekli et al., 2010b), used to generate functional compositions in terms of CP-Nets. Since the compositions can contain serial and concurrent mapped functions, we provide an algorithm that we develop based on CP-Nets' properties for discovering and generating processing sequences simultaneously for serial and concurrent compositions. To validate our approach, we develop a prototype for the XA2C framework and use it to test our processing sequence generation algorithm with different scenarios.

The rest of this paper is organized as follows. The first section presents the related work. Section 2 discusses the XA2C framework with the XCDL language and the

process sequence generator algorithm. Section 3 presents the prototype and evaluates the algorithm. And finally, we conclude and state some future works.

Related Work

XML manipulation approaches have been argued since XML has emerged. They were initially discussed in technical terms from the point view of experts in the field. Most recently, these approaches have evolved to reach the needs of non-expert programmers due to the widespread of XML nowadays. 3 main approaches have emerged, addressing the XML manipulation issue from different angles (i.e., expressiveness, human interactions, expertise etc.), XML alteration/adaptation, Mashups, and XML visual languages.

XML Alteration/Adaptation

The alteration/adaptation field of control resides in modifying and adapting the XML data to satisfy the needs of a user(s). In this case, researchers have been developing different solutions with separate scopes such as encryption and digital signatures, filtering, adaptation and information extraction.

XML encryption and digital signatures were mainly introduced to secure XML data communications and make sure that the data integrity remains intact between end users. They are used to obfuscate XML data and authenticate XML users.

XML encryption and signature were standardized by the W3C (World Wide Web Consortium). Other formalizations were established allowing both encryption and signature in the same language such as in (Hwang and Chang, 2004). Encryption and signature are applicable on 2 levels: document and element-wise. XML encryption

and signature constitute a small part of XML control as viewed in our research. It can be categorized in either the security field of control or the modification/adaptation field of control depending on its use. This technique still lacks the ability to allow a granular encryption or signature of the element content data.

XML filtering has been and still is one of the main fields that researchers have been developing in order to apply some control and adaptation of XML data to user specifications. XML filtering can be described as, given a set of twig patterns, retrieve the data corresponding to these patterns in an input XML document or data. XML filtering results in a granular selection of XML data. Its granularity degree depends on the filter applied. Several filtering techniques have been developed based on either XPath expressions or a subset of XQuery. Some of these main techniques are XFilter (Altinel and Franklin, 2000), YFilter (Diao et al., 2003), QFilter (Luo et al., 2004), PFilter (Byun et al., 2007) and AFilter (Candan et al., 2006). These techniques have been evolving using mainly deterministic finite automata (DFA) and non-deterministic finite automata (NFA) for either structural matching or value based-predicates. The supported range of value based predicates has evolved from equality operators to non equality operators, Boolean operators (AND/OR) and finally the special matching operator "%" processed similarly as the LIKE operator in SQL. Basically XML filters use XQueries or XPath expressions and transform them into DFAs and NFAs, thus defining the twig patterns specified by users in order to find specific XML data. XML filtering is a selection technique and does not involve XML data modification and therefore does not satisfy our objectives.

Several researches have been conducted concerning **XML content adaptation**, mostly on XML document describing multimedia content such as XHTML, SMIL (Lemlouma and Layaïda, 2003), SVG (Pellan and Concolato, 2008). There were some researches conducted on adapting XML documents and transforming them to other XML documents to satisfy a certain objective based on the XSLT standard (W3C, 1999). Due to the complexity found in XSLT this approach was categorized by users as complicated and limited to the actions allowed by the XSLT language. Yet the main goal of XML adaptation has been so far to adapt multimedia content such as images, audio and video sequences to be viewed on appropriate terminals (e.g., portable multimedia devices, mobile phones and HD displays). The adaptations are made mostly in terms of resolutions, aspect ratios and size in correspondence to the terminals displaying the data and their specifications. The adaptation mechanism in multimedia content adaptation is normally based on the properties of the document containing the data which has a well known structure and is well defined to contain multimedia data such as in SMIL or SVG (Pellan and Concolato, 2008, Lemlouma and Layaïda, 2003). XML adaptation remains somewhat complex and focalized on multimedia based documents.

Data extraction and modification is one of our main goals for controlling XML data. Several solutions exist for **data extraction or IE** (information extraction) based on the usage of wrappers. These solutions are mainly aiming at IE from web pages instead of XML files and storing the extracted info into a database or XML files. Some of them are IEPAD (Chang and Lui, 2001), Nodose (Adelberg, 1998) and ROADRUNNER (Crescenzi et al., 2002). These approaches mainly rely on visual

information which are either defined by the browser or the user. No standardized approach exists yet. They are viewed as applications or tools which learn from examples given by the user in order to generate IE rules. Most of these approaches view web pages as trees which are consider faster in data extraction. Nonetheless, these approaches are inadequate or insufficient in our research due to their lack of formalism, do not directly aim at XML data but web pages instead and are limited to the tools used for data transformation which are user-based and not following any unified existing models or standards.

Tab.I: Scope and Data types of existing alteration/adaptation control techniques

Techniques	**Scope**	**XML data type**
Obfuscation	Document and element-wise obfuscation	All XML data types
Filtering	Granular selection of XML data	All XML data types
Adaptation	XML-based multimedia data modifications to render it conform to an alien system (e.g., PDAs).	Mainly multimedia XML data
IE	Data Extraction based on rules and storage in a DB, XML files or others	Mainly Web Pages

To summarize, instead of working separately on each of the precedent alteration/adaptation approaches and having to manually adapt them together, as shown in Table I, there is a need for a framework with a unified language allowing simultaneously the expression of structural and content filtering, adaptation, granular encryption similarity comparisons and others, regardless to the type of XML data.

Mashups

Mashup is a new application development approach that allows users to aggregate multiple services, each serving its own purpose, to create a service that serves a new purpose. Mashups are built on the idea of reusing and combining existing services. They are mainly designed for data circulating on the web. Their objective is to target non-expert users; therefore a graphical interface is generally offered to the user to express most operations. Mashup applications (Lorenzo et al., 2009) can include Mashups using maps (i.e., Google maps and Yahoo map3), multimedia content (i.e., YouTube and Flicker videos), e-commerce services (i.e., amazon.com and ebay.com) and news feeds (i.e., RSS and ATOM). The latter is the focus of most emerging Mashup tools nowadays.

To the best of our knowledge, no tool yet provides information regarding the analysis of the performances. All the tools are supposed to target non-expert users, but a programming knowledge is usually required depending on each tool.

Several tools have emerged such as Damia (Simmen et al., 2008), Yahoo Pipes (Lorenzo et al., 2009), Popfly (Loton, 2008), Apatar (Lorenzo et al., 2009) and MashMaker (Ennals and Garofalakis, 2007). Damia and Yahoo Pipes are mainly designed to manipulate Data Feeds such as RSS feeds. Popfly is used to visualize data associated to social networks such as Flicker and Facebook. Popfly is a framework for creating web pages containing dynamic and rich visualizations of structured data retrieved from the web through REST web services. Apatar helps users join and aggregate data such as MySQL, Oracle and others with the web through REST web services. MashMaker is used for editing, querying and manipulating data from web

pages. Its goal is to suggest to the user some enhancements, if available, for the visited web pages.

To summarize, existing Mashup tools are (i) mainly designed to handle online Web data which is restrictive in several scenarios since by doing this, user's data, generally available on desktops cannot be accessed and used, (ii) not specifically designed for XML data manipulation and therefore do not provide XML specific operations for querying, updating and modifying all types of XML data and, (iii) going towards functional compositions (i.e., Damia and Yahoo Pipes) which allows them to increase their expressiveness in comparison with the tools following the query by example paradigm (Lorenzo et al., 2009). The latter have limited operations and are considered more complex for non expert users due to the fact that some knowledge is required in querying data.

Visual Languages for XML

Since the emerging of the XML standard and its widespread beyond the computer domain, researchers have been trying to provide visual languages allowing the manipulation of XML data. These visual languages are mainly extensions of existing approaches such as XML query languages and transformation languages. Their main contribution is to allow non expert users to extract sensitive data from XML document and restructure the output document.

Several languages have been developed over the years such as Xing (Erwig, 2000), XML-GL (Ceri et al., 2000), XQBE (Braga et al., 2005) and VXT (Pietriga et al., 2001). On one hand, Xing and XML-GL were developed before XQuery was

standardized and took the SQL querying approach by following the 3 main components of a regular query, selecting, filtering and restructuring the data. XQBE was developed after XQuery and is based on it. Its expressiveness is greater than previous approaches whereas it allows the creation of complex queries containing aggregation functions, ordering results and negation expressions. Nonetheless, its expressiveness is limited to data extraction and query reconstruction in XQuery and does not include textual data manipulation operations such as value modification, insertion and deletion. VXT was based on XSLT (W3C, 1999) which is mainly used for XML data restructuring and not textual data manipulation.

From a visual perspective, all of these approaches followed the same pattern, dividing their workspace into 2 main sections, left and right. The left section constitutes the source file with the extraction rules. As for the right section, it defines the structure of the output file. The sections are mapped together as shown in Figure 2.

Tab.II: Properties of Mashups and XML oriented languages

Properties	**Mashups**	**XML Visual Languages**
XML specific	**No**	**Yes**
Manipulate online data	**Yes**	**Yes**
Manipulate desktop data	**No**	**Yes**
Expressiveness	**High**	**Low**
Based on Formal languages	**No**	**Yes**
Functional Composition	**Yes**	**No**
Composition-based functions	**No**	**No**

Extending functions	**Dependent on the tool**	**Limited**

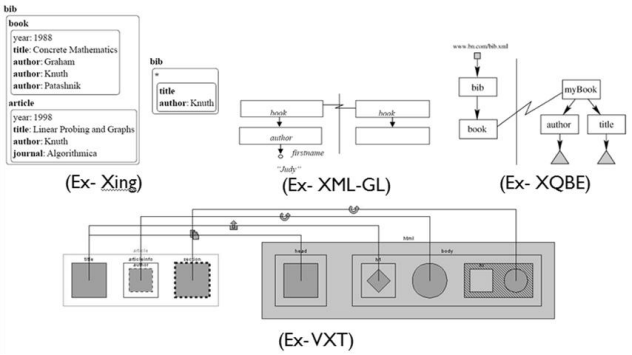


Fig.2: Examples of existing visual languages for XML

To summarize, existing visual languages successfully bridged the gap between the complexities of XML data querying and non expert users but were limited only to data extraction, filtering and restructuring. So mainly they provided non expert programmers with the ability to create XML structural transformations along with data extraction and filtering but did not deal with the XML value manipulations.

Table II summarizes the different criteria of the Mashups and XML oriented visual languages.

Preliminaries and Definitions

In this paper, we present the XA2C framework based on Colored Petri Nets (CP-Nets) and 2 of their main properties: (i) the incidence matrix and (ii) transition firing rule. As stated in (Jensen, 1994, Murata, 1989), a Petri Net is foremostly a mathematical description, but it is also a visual or graphical representation of a system. Petri nets allow the definition of the state and behavior of a language simultaneously, in contrast with most specification languages. They provide an

explicit description of both the states and the actions. Petri nets were mainly designed as a graphical and mathematical tool for describing and studying information processing systems, with concurrent, asynchronous, distributed, parallel, non deterministic and stochastic behaviors. They consist of a number of places and transitions with tokens distributed over places. Arcs are used to connect transitions and places. When every input place of a transition contains a token, the transition is enabled and may fire. The result of firing a transition is that a token from every input place is consumed and a token is placed into every output place.

CP-nets have been developed, from being a promising theoretical model, to being a full-fledged language for the design, specification, simulation, validation and implementation of large software systems.

In a CP-Net:

- The states are represented by means of places (drawn as ellipses)
- The actions are represented by means of transitions (drawn as rectangles)
- An incoming arc indicates that the transition may remove tokens from the corresponding place while an outgoing arc indicates that the transition may add tokens
- The exact number of tokens and their data values are determined by arc expressions (positioned next to the arcs)
- The data types are referred to as color sets
- A transition has an expression guard (with variables) attached to it defining its operation.

A CP-Net is formally defined as follows:

Definition 1-*Colored Petri Net or CP-net:* *it is an 8-tuple represented as:*

CP-Net = (Σ, P, T, A, C, G, E, I) *where:*

- *Σ is a finite set of non-empty types also called color sets*
- *P is a finite set of places*
- *T is a finite set of transitions*
- *A is a finite set of arcs such that:*
  - $P \cap T = P \cap A = T \cap A = \emptyset$
- *C is a color function. It is defined from P into Σ*
- *G is a guard function. It is defined from T into expressions such that:*
  - $\forall t \in T: [Type(G(t)) \subseteq \Sigma]$
- *E is an arc expression function. It is defined from A into expressions such that:*
  - $\forall a \in A: [Type(E(a)) = C(p) \wedge Type(Var(E(a))) \subseteq \Sigma]$

  where p is the input place of a
- *I is an initialization function. It is defined from P into closed expressions such that:*
  - $\forall p \in P: [Type(I(p)) = C(p)]$

The types of a variable *v* and an expression *expr* are denoted *Type(v)* and *Type(expr)* respectively. *Var(expr)* designates the variables of an expression *expr*. An example of a CP-Net is depicted in Figure 3. This CP-Net has 3 places: two of them have a type `Int×String`, and one has a type `Int`. The transition takes one token of the pair type and one of the integer type, and produces one token of the pair type.

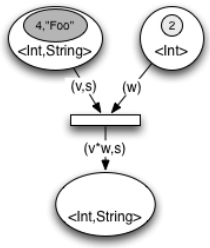


Fig.3: An example of a CP-Net

In this paper we are particularly interested in 2 main properties of CP-Nets, the Incidence Matrix and the Transition Firing Rule.

Definition 2-*Incidence matrix A*: *it is defined for a CP-Nets N with m transitions and n places as:*

$$\boldsymbol{A = [a_{ij}]}$$ *an n×m matrix of integers where:*

- $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$ where
  - $a_{ij}^{+} = w(i,j)$ is the weight of the arc from transition *i* to its output place *j*
  - $a_{ij}^{-} = w(i,j)$ is the weight of the arc to transition *i* from its input place *j*

a_{ij}^{+}, a_{ij}^{-} *and* a_{ij} *represent the number of tokens removed, added, and changed in place j when transition i fires once.*

Table III shows the Incidence Matrix of the CP-Net in Figure 3 which identifies p_1 and p_2 as input places of transition t and p_3 its output place.

Tab.III: Incidence Matrix of CP-Net in Figure 3

A=

	t
p_1	-1
p_2	-1
p_3	1

Definition 3-*Firing Rule*: *it is the conditions for a transition to fire and is defined as:*

t is enabled if M(p) ≥ w(p,t) for all input p to t *where:*

- *A transition "t" is enabled if each input place "p" of "t" is marked with at least "w(p,t)", where "w(p,t)" is the weight of the arc from "p" to "t"*
- *An enabled transition t may or may not fire (depending on whether event takes place or not)*
- *A firing of an enabled transition t removes w(p,t) token from each input place p to t and adds w(t,p) tokens to each output place p of t*

Next we present our approach by defining the architecture of the XA2C Framework, giving a brief introduction to our visual composition language, the XCDL language, and then discussing our algorithm for deriving the concurrent execution sequences of the resulting composition.

Our Approach

As mentioned previously, the purpose of our research is to provide non-expert and expert programmers with means to compose XML oriented manipulation operations, thus altering and adapting XML based data to their needs. The approach needs to be

both generic to all XML data (text-centric and data-centric) and needs to be well founded, in order to allow it to be portable and reusable in different domains (i.e., Mashups, XML adaptation/alteration platforms, XML transformation and extraction, textual data manipulations, etc.).

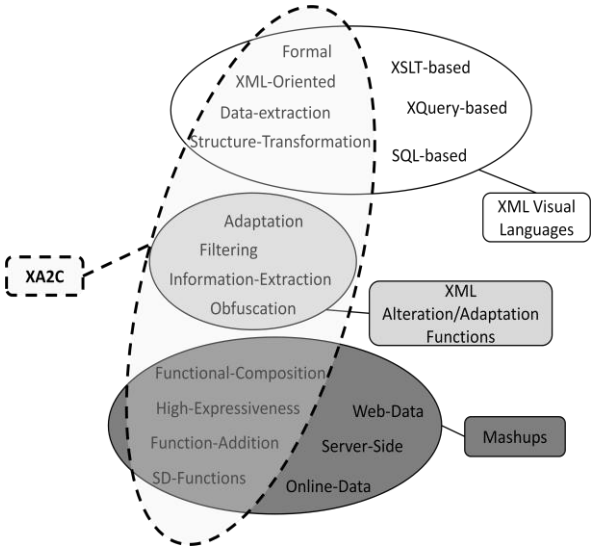


Fig.4: XA2C approach

As stated in the introduction, there has been no existing formal and generic approaches answering such matters, nonetheless, several approaches have emerged undertaking different aspects of our research such as, (i) Mashups, which are neither formalized nor XML specific, are being oriented towards functional compositions and scope non expert programmers, (ii) XML visual languages, while they are formalized and XML specific, they provide only XML data extraction and structural transformations but no XML data manipulations, mainly text-centric based, and (iii) XML alteration/adaptation techniques are dispersed from one another resolving each

to a different objective (e.g., filtering, data extraction, etc.) and require expertise in their appliances.

As shown in Figure 4, our approach is based on a combined spirit of both Mashups and XML visual languages. On one hand, it has a similar architecture to Mashups that renders the framework flexible thanks to its modular aspect and is based on functional compositions which are considered simpler to use than query by example techniques. On the other hand, it defines formally a visual composition language and separates the inputs and outputs to source and destination structures, thus making the framework XML-oriented. Similar to XML-oriented visual languages, the approach targets both expert and non-expert programmers.

The visual composition language defined in the XA2C can be adapted to any composition based Mashup tool or visual functional composition tool. Nevertheless, our language is XML-oriented and generic to all types of XML data (documents and fragments, grammar-based and user-based). In addition, it is based on CP-Nets allowing us to provide information regarding performance analysis and error handling which is not the case in current Mashups. To render our approach flexible, the XA2C framework is defined as a modular architecture as shown in Figure 5.

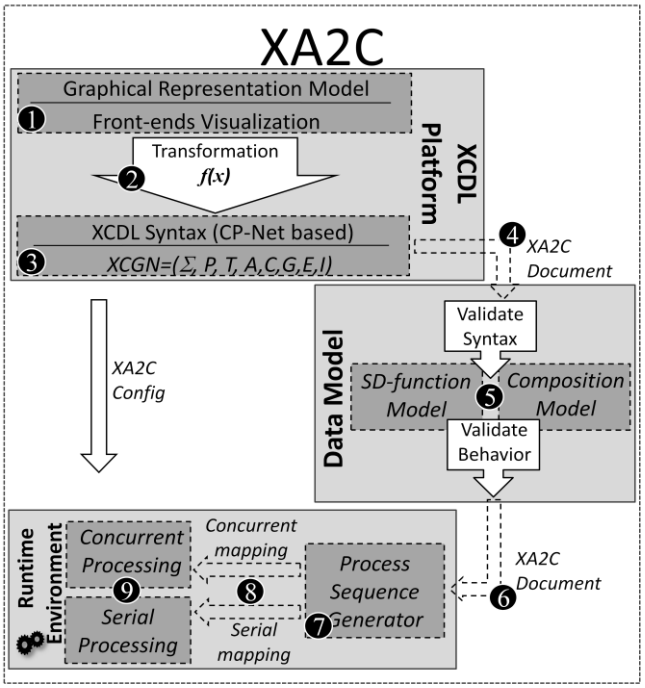


Fig.5: Architecture of the XA2C Framework

Our framework is composed of 3 main modules:

- The **XCDL Platform** allows the definition of the XCDL language providing non-expert and expert programmers with the means to define their manipulation operations. The language mainly allows users to define their functions from offline or online libraries and create manipulation operations by composing these functions using mapping operators. The XCDL is defined as a visual functional composition language based on the graphical representations and algebraic grammar of CP-nets. Thus, rendering the language extensible and generic (adaptable to different data types), and allowing the expression of true concurrency along with serial compositions. As a user defines a new function or

modifies a composition (adding, removing, replacing functions), the syntax is transmitted to the data model module to be continuously validated.

- The **Data Model** contains the internal data models of the XA2C which are based on the same grammar used to define the syntax of the XCDL language (naturally based on CP-Nets). We define 2 internal data models: (i) "SD-function (System-Defined function) Data Model" for validating the components of the language, in this case to validate the defined functions in our system, and (ii) "Composition Data Model" used to validate the compositions. The validation process is event-based, any modification to the language components or to a composition such as additions, removals or editions trigger the validation process.
- The **Runtime Environment** defines the execution environment of the resulting compositions of the XCDL language. This module contains 3 main components: (i) the "Process Sequence Generator" used to validate the behavioral aspect of the composition (e.g., makes sure there are no open loops, no loose ends, etc.) and generates 2 processing sequences, a concurrent and a serial one to be transmitted respectively to the Concurrent and Serial Processing components for execution.

  (ii) "Serial Processing" allowing a sequential execution of the "Serial Sequence" provided by the data model. It is more suitable for single processor equipped machines as it will not take advantage of a multi-processing unit.

  (iii) "Concurrent Processing" allowing the execution in a concurrent manner of the "Concurrent Sequence" generated from the data model. It is imperative to note that this type of processing is most suitable for machines well-equipped for multi-processing tasks (e.g., dual processors machines). Due to the lack of space,

the Serial and Concurrent Processing components are not detailed in this paper, but will be discussed in future studies.

In the next section, we briefly discuss each of the 3 modules.

XCDL Platform

The XCDL is a visual functional composition language based on *SD-functions* (*System-Defined functions*) and is XML-oriented. The language is rendered generic, extensible and user friendly by respecting the following properties: (i) simplicity, (ii) expressiveness, (iii) flexibility, (iv) scalability, and (v) adaptability. These properties are satisfied by defining the language as a visual one and basing its syntax on a grammar defined in CP-Nets (cf. Definition 4) and therefore retains their properties such as Petri Net firing rule and Incidence matrix.

Definition 4-*XCGN* *(standing for XML oriented Composition Grammar Net): it represents the grammar of the XCDL which is compliant to CP-Nets. It is defined as:*

$$XCGN = (\Sigma, P, T, A, C, G, E, I) \text{ where:}$$

- Σ *is a set of data types available in the XCDL*
  - *The XCDL defines 6 main data types,* $\Sigma = \{Char, String, Integer, Double, Boolean, XML\text{-}Node\}$ *where Char, String, Integer, Double and Boolean designate the standard types of the same name. XML-Node defines a super-type designating an XML component (cf. definition 5)*
- *P is a finite set of places defining the input and output states of the functions used in the XCDL*
- *T is a finite set of transitions representing the behavior of the XCDL functions and operators*
- $A \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs associating input places to transitions and vice versa*
  - $\forall a \in A$: *a.p and a.t denote the place and transition linked to arc a*
- $C: P \rightarrow \Sigma$ *is the function associating a color to each place*
- $G: T \rightarrow S$ *is the function associating an SD function to a transition where:*
  - *S is the set of SD-functions, which are operations performed by functions identified in the development platform's libraries (e.g., concat(string,string))*
- $E: A \rightarrow Expr$ *is the function associating an expression* $expr \in Expr$ *to an arc such that:*
  - $\forall a \in A: Type(E(a)) = C(a.p)$
- $I: P \rightarrow Value$ *is the function associating initial values from Value to the I/O places such that:*
  - $\forall p \in P, \forall v \in Value : [Type(I(p)) = C(p) \wedge Type(v) \in \Sigma]$

Definition 5-*XML-Node:* *it is a super type designating an XML Component. It has 3 main sub-types as:*

XML-Node $\in$ ***{XML-Node:Element, XML-Node:Attribute*** *and* ***XML-Node:Text}*** *where:*

- *XML-Node:Element defines the type XML Element*
- *XML-Node:Attribute defines the type XML Attribute*
- *XML-Node:Text define the type XML Element/Attribute Value*

We denote by *SD-functions*, functions which will be identified in the language environment. These *SD-functions* can be provided by offline libraries (e.g., DLL/JAR files) or online libraries (e.g., Web service).

XCDL is divided into 2 main parts:

- The Inputs/Outputs (I/O)
- The *SD-functions* and the composition which constitute the XCDL Core.

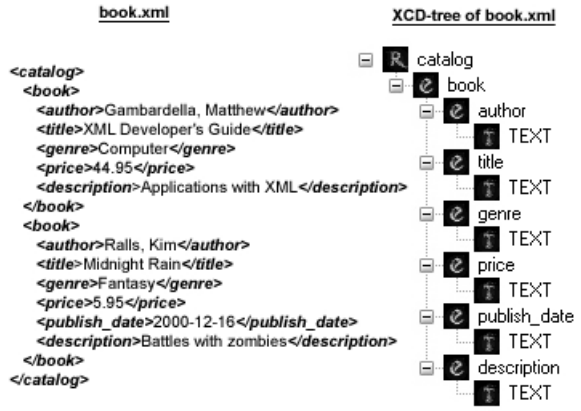


Fig.6: XML document to XCD-tree example

The I/O are defined as XML Content Description trees (Tekli et al., 2010c) (XCD-trees) which are ordered labeled trees summarizing the structure of XML documents or XML fragments, or representing a DTD or an XML schema, in forms of tree views as shown in Figure 6.

SD-functions are defined each as a CP-Net with the inputs and outputs defined as places and represented graphically as circles filled with a single color each defining their types (e.g., String, Integer, etc.). It is important to note that a function can have one or multiple inputs but only one output. The operation of the function itself is represented in a transition which transforms the inputs to the output. Graphically, it is represented as a rectangle with an image embedded inside it describing the operation. Input and output places are linked to the transition via arcs represented by direct lines. Several sample functions are shown in Figure 7.

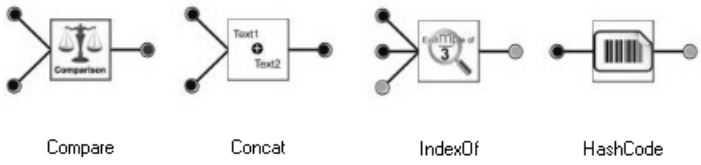


Fig.7: Sample functions defined in XCDL

The composition is also based on CP-Nets. It is defined by a sequential mapping between the output and an input of instances of *SD-functions*. The functions are dragged and dropped, and then linked together with a *Sequence Operator "➔"* which is represented by a dashed line between the output of a function and an input of another, having the same color as shown in Figure 8.

As a result, on one hand, a composition might be a serial one meaning that all the functions are linked sequentially and to each function one and only one function can be mapped as illustrated in Figure 8.a. In this case, the sequential operator is enough. However, the composition might contain concurrency, as in, several functions can be mapped to a single one as depicted in Figure 8.b. In this case, we introduce an

abstract operator, the *Concurrency Operator* "//", in order to indicate the functions are concurrent.

As shown in Figure 8, we define 2 main types of compositions, a Serial Composition "$\boldsymbol{SC} = \prod_{i=0}^{n} \boldsymbol{SDF}_i \rightarrow_i$" (cf. Definition 6) and a Concurrent Composition "$\boldsymbol{CC} = \prod_{i=0}^{n} (\boldsymbol{SDF}_i \rightarrow_i \boldsymbol{SDF}_{n+1})$ //" (cf. Definition 7).

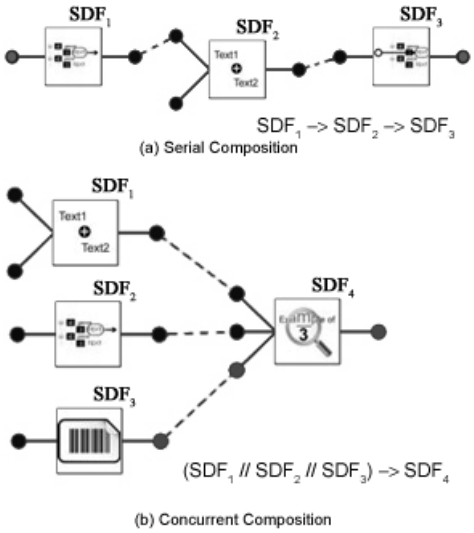


Figure 8: Serial and Concurrent Compositions

In the XCDL Core, we separate the composition to a Serial Composition mapping sequentially several instances of *SD-functions* and a Concurrent Composition, mapping several instances of *SD-functions* sequentially to a single instance of *SD-function*. Figure 8.a and 8.b illustrate respectively a Serial Composition and a Concurrent Composition.

Definition 6-*SC:* *it is a Serial Composition,* $SC = \prod_{i=0}^{n} SDF_i \rightarrow_i$, *linking sequentially* ***n*** *instances of SD-functions using* ***n-1*** *instances of Sequence operators and is compliant to a CP-Net. It is defined as:*

$\boldsymbol{SC} = \prod_{i=0}^{n} \boldsymbol{SDF}_i \rightarrow_i = (\boldsymbol{\Sigma, P, T, A, C, G, E, I})$ *where:*

- SDF_i *is a SD-function where:*

$\forall i,j \in [0,n]$, $SDF_i \neq SDF_j$ *for* $i \neq j$

- $\rightarrow_i$ *is a Seque*$_n$*ce ope*$_r$*ator where:*
  - $\rightarrow_i.\Sigma \subseteq SDF_i.\Sigma$
  - $\rightarrow_i.P_{In} = SDF_i.P_{Out}$ *and* $\rightarrow_i.P_{Out} \in SDF_{i+1}.P_{In}$
  - $\rightarrow_n$ *= (Ø, Ø, Ø, Ø, C, G, E, I) in an empty CP-Net*
- $\Sigma = \bigcup_{i=0}^{n} SDF_i.\Sigma$
- $P = P_{In} \cup P_{Out}$ *where* $P_{In} = \bigcup_{i=0}^{n} SDF_i.P_{In}$ *and* $P_{Out} = \bigcup_{i=0}^{n} SDF_i.P_{Out}$
- $T = \bigcup_{i=0}^{n}(SDF_i.T \cup \rightarrow_i.T)$
- $A = \bigcup_{i=0}^{n}(SDF_i.A \cup \rightarrow_i.A)$
- *C:P* $\rightarrow \Sigma$ *is the function associating a color to each place where C = SD-function.C*
- *G: is a function over T where*

$$\forall t \in T, G(t) = \begin{cases} SDF_i.G(t), & t \in \bigcup_{i=0}^{n} SDF_i.T \\ \rightarrow_i.G(t), & t \in \bigcup_{i=0}^{n} \rightarrow_i.T \end{cases}$$

- *E:A* $\rightarrow$ *Expr is the function associating an expression expr to an arc a where E = SD-function.E*
- *I:*P_{In} $\rightarrow$ *Value is the function associating initial values to the Input places, I = SD-function.I*

Definition 7-*CC:* *it is a Concurrent Composition,* $CC = \prod_{i=0}^{n}(SDF_i \rightarrow_i SDF_{n+1})$ *//, linking* ***n*** *instances of SD-functions using* ***n*** *instances of Sequence operators concurrently to an instance of SD-function and is compliant to a CP-Net. It is defined as:*

$\boldsymbol{CC = \prod_{i=0}^{n}(SDF_i \rightarrow_i SDF_{n+1})//=}$ ***(Σ, P, T, A, C, G, E, I)*** *where:*

- SDF_i *and* SDF_{n+1} *is a SD-function where:*
  - $\forall i \in [0,n+1]$ *and* $\forall j \in [0,n+1]$, $SDF_i \neq SDF_j$ *for* $i \neq j$
- $\rightarrow_i$ *is a Sequence operator where:*
  - $\rightarrow_i.\Sigma \subseteq SDF_i.\Sigma$
  - $\rightarrow_i.P_{In} = SDF_i.P_{Out}$ *and* $\rightarrow_i.P_{Out} \in SDF_{n+1}.P_{In}$
- $\Sigma = \bigcup_{i=0}^{n+1} SDF_i.\Sigma$
- $P = P_{In} \cup P_{Out}$ *where* $P_{In} = \bigcup_{i=0}^{n+1} SDF_i.P_{In}$ *and* $P_{Out} = \bigcup_{i=0}^{n+1} SDF_i.P_{Out}$
- $T = \bigcup_{i=0}^{n}(SDF_i.T \cup \rightarrow_i.T) \cup SDF_{n+1}.T$
- $A = \bigcup_{i=0}^{n}(SDF_i.A \cup \rightarrow_i.A) \cup SDF_{n+1}.A$
- *C:P* $\rightarrow \Sigma$ *is the function associating a color to each place where C = SD-function.C*
- *G: is a function over T where*

$$\forall t \in T, G(t) = \begin{cases} SDF_i.G(t), & t \in \bigcup_{i=0}^{n+1} SDF_i.T \\ \rightarrow_i.G(t), & t \in \bigcup_{i=0}^{n} \rightarrow_i.T \end{cases}$$

- *E:A* $\rightarrow$ *Expr is the function associating an expression expr to an arc a where E = SD-function.E*
- *I:*P_{In} $\rightarrow$ *Value is the function associating initial values to the Input places, I = SD-function.I*

The syntax of the *SD-functions*, *Sequence Operator*, *Concurrency operator*, are all based on the grammar defined by CP-Nets' algebra as discussed in detail in (Tekli et al., 2010b). Figure 9 shows an illustration of a combination of a serial and concurrent composition.

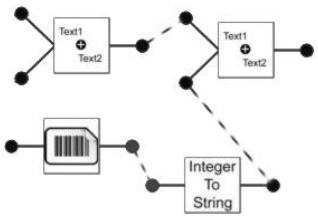
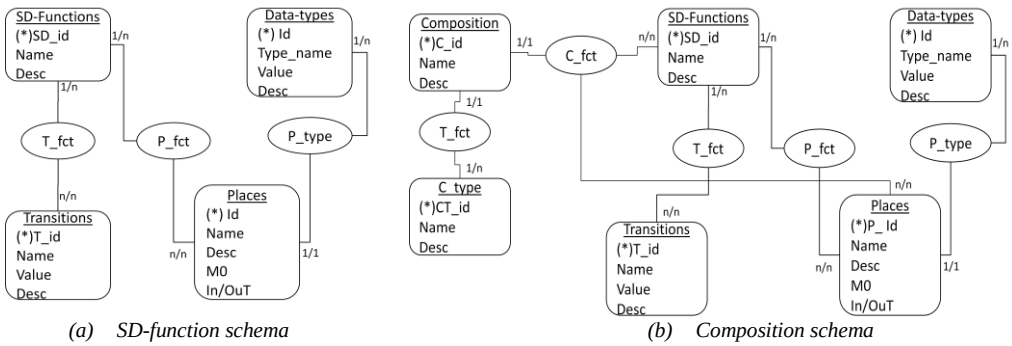


Fig.9: Composition example in XCDL

In this example, we can see that SDF1 is concurrently mapped to SDF2 with a serial composition of SDF3 and SDF4. In this case the composition is expressed as follows: "$C = (SDF_1 // (SDF_3 \rightarrow SDF_4)) \rightarrow SDF_2$" and the resulting composition is a CP-Net compliant to the XCGN and transmitted to the Data Model for validation.

Data Model

As stated earlier, this module is used to validate the syntax and behavior of the composition expressed in the XCDL platform. The main purpose is to ensure that the composition is compliant to our grammar. In Figure 5, we can see that this module contains 2 main components: (i) SD-function Model and (ii) Composition Model. They define the internal data model of the XA2C.



(a) SD-function schema *(b) Composition schema*

Fig.10: Relational schemas compliant with XCGN

The SD-function Model shown in Figure 10.a is defined as a relational schema representing SD-functions as CP-nets. This schema is used to validate SD-functions before they are stored in the system.

As an example, consider the *SD-function* "Concat" shown in Figure 7. This function is defined as follows:

Concat = (Σ, P, T, A, C, G, E, I) *where*:

- $\Sigma = \{String\}$
- $P = P_{In} \cup P_{Out} = \{In_Str_1, In_Str_2\} \cup \{Out_Str\}$
- $T = \{Concat\}$
- $A \subseteq (P_{In} \times \{t\}) \cup (\{t\} \times P_{Out})$
- $C: P \rightarrow \Sigma$ *where* $C(In_Str_1) = C(In_Str_2) = C(Out_Str) = String$
- $G: \{t\} \rightarrow S$ *where* $G(Concat) = String_functions.Concat$ *and* $Type(G(Concat)) = C(Out_Str) = String$ *where String_functions is the DLL containing String manipulation functions.*
- $E: A \rightarrow Expr$ *is the function associating an expression* $expr \in Expr$ *to* $a \in A$*:*
    - $Expr = \{M(In_Str_1), M(In_Str_2), G(Concat)\}$ *is a set of expressions where:*

$$\forall expr \in Expr:\ expr = \begin{cases} M(a.p)\ if\ a.p \neq p_{Out} \\ G(a.t)\ otherwise \end{cases}$$

- $I: P_{In} \rightarrow Value$ *where* $I(In_str_1) = I(In_str_2) = ""$

The "Concat" *SD-function* is validated through the SD-function model which will allow it then to be stored as a CP-Net in a XML based file.

The Composition Model shown in Figure 10.b is also defined as a relational schema which is used to validate the syntax of the composition before storing it as a CP-Net in a XML based file and transmitting it to the Processing Sequence Generator in the Runtime Environment module for execution sequence discovery and generation.

Runtime Environment

As stated in the previous section, the XCDL is based on the XCGN, a grammar based on CP-Nets, and the resulting composition is a CP-Net. The Process Sequence Generator is used to generate 2 execution sequences, serial and concurrent sequences which specify the order in which the composed functions can be executed.

The Concurrent Sequence specifies different concurrency levels (CL) which must be executed in an sequential manner from CL_0 to CL_n where n is the last CL. Each CL contains 1 or several functions which can be executed in a concurrent manner (parallel or serial).

The Serial Sequence defines the execution of the functions in a serial manner where each of the functions in the composition will have a unique order in which it can be executed ranging from 0 to m-1, m is the number of functions used in the composition.

In order to generate both sequences, we provide an algorithm based on the Incidence Matrix (Murata, 1989) of CP-Nets (cf. Definition 2).

Before we give the algorithm, we present the hypothesis defining the background on which the algorithm is based upon.

Hypothesis:

Based on the XCDL syntax, defined in the XA2C platform, the resulting composition is also defined as a CP-Net based on the XGCN and respects the following main properties:

- Each place can contain one and only one token
- A token can be added either through an initial marking provided by the user or an XCD-tree node or through a fired transition
- All arcs are weighted with the value 1
- A transition is enabled once each of its input places contains at least one token

- A fired transition clears its input places of all tokens and generates one and only one token in each of its output places.

Based on these properties, we define our algorithm for simultaneously discovering and generating a serial and concurrent function processing sequence. The processing sequence is stored in a 2 dimensional matrix (called PP for Parallel Processing) where each line represents the concurrent level of execution and each column represents a transition (a *SD-function*).

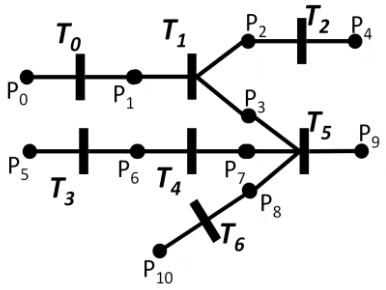


Fig.11: CPN_1, an Example of a CP-Net resulting from the XCDL

Consider the composition CPN_1 in Figure 11, Table IV represents its PP matrix. The PP matrix shows that we have 3 CLs which must be executed sequentially and orderly from CL_0 to CL_2 (e.g., T1 and T4 are enabled once T0, T3 and T6 have fired). All transitions in a CL can be executed simultaneously in parallel. As shown in Table IV, each transition corresponding to a CL is assigned a number. This number represents the sequence order in which a transition should fire in Serial Processing mode (e.g., in Table IV T0, T3, T6, T1, T4, T2 and T5 will be executed sequentially in Serial Processing mode).

Tab.IV: PP matrix of the CP-Net in Figure 11

CL/T	T_0	T_1	T_2	T_3	T_4	T_5	T_6
CL_0	0			1			2
CL_1		3			4		
CL_2			5			6	

We present next the skeleton of the algorithm followed by the algorithm generating the PP matrix.

Algorithm skeleton:

The algorithm contains 2 loop steps:

- *Step 1 (lines 1-18):* For each place in A, check if the initial value is of type "XCD node" or "user" (in other terms, checks if the place is a source place), if so, then for each transition in A check if the corresponding place is an input to the transition. If the place is found to be an input then clear its value from A and check if the transition is enabled. If it is enabled and *PP* does not contain a value in the corresponding transition column then add the value of *m* in *PP(j,n)* where j is the index of the enabled transition and increment *m* by 1. If the transition is enabled and *PP* already contains a value in the corresponding transition column, then report an error in the composition and exit the algorithm.
- *Step 2 (lines 19-42):* While |PP| < T.num, for each transition in PP on CL_{n-1}, clear all its output places and if they are input places to other transitions, clear them as well from A, then check if their corresponding transitions are enabled, if so then check that they were not already added to PP and add them in the corresponding

transition line on the CL_n, otherwise, return an error in the composition and exit the algorithm.

The formal algorithm is presented here below.

Algorithm's Pseudo-Code:

```
    Inputs:
Integer A(,)    // A is the Incidence matrix
String T(),P()  // T is the Transitions matrix
                // P is the Places matrix
    Outputs:
Integer PP(,)   // PP is the Parallel Processing matrix
    Variables:
Var PP(,) as Integer(T.num,1)
Var m, n as Integer = 0
// m is the sequence number of the next transition
// n is the current level number of the parallel processing
    Begin:
    // step 1
1.  for i = 0 to (P.num - 1)
2.    if (P_type(i) = "in xcd") | (P_type(i) = "user") then
3.      for j = 0 to (T.num - 1)
4.        if A(i,j) = -1 then
5.          A(i,j) = 0
6.          if T_enabled(i,j) then
7.            if not (PP.contains(get_t(out_p))) then
8.              PP(j,n) = m
9.              m = m+1
10.           else
11.             Error("Composition Error")
12.             Exit
13.           end if
14.         end if
15.       end if
16.     end for
17.   end if
18. end for

    // step 2
19. while (m < T.num)
20.   for i = 0 to (T.num - 1)
21.     if PP(i,n) not Null then
22.       t=T(i)
23.       for each out_p in A.outputs(t)()
24.         out_p = 0
25.         for each in_p in A.inputs(get_t(out_p))()
26.           if in_p = out_p then
27.           in_p = 0
```

```
28.         end for
29.         if get_t(out_p).enabled then
30.           if not (PP.contains(get_t(out_p))) then
31.             PP(get_t(out_p),n) = m
32.           else
33.             Error("Composition Error")
34.             Exit
35.           end if
36.         end if
37.       end for
38.     end if
39.   end for
40.   n = n + 1
41. end while
    End
```

In case of a valid composition, the Process Sequence Generator must ensure that (i) All transitions are present in *PP* and each transition is present once and only once, (ii) After attending the i^{th} level, if all transitions in level *i* fire then all transitions in level *i+1* are enabled and (iii) All transitions in level *i* can be executed in parallel.

Therefore, to prove the correctness of our algorithm, we must prove the following 3 lemmas.

Lemma 1. If ($\exists$ PP) Then ($t_i \neq t_j,\ \forall i,j \in N,\ i,j < T.num\ and\ i \neq j$)

Proof. Before populating the PP matrix, whether in *loop step 1* or *2*, the algorithm checks each time at line 7 and 30 respectively if the added transition already exists, if so then the execution is interrupted and PP is not generated and:

$$\text{If } \left((\forall i,j \in N \text{ and } i,j < T.num\ and\ i \neq j),\ \exists(t_i = t_j)\right) \text{ then } (\nexists PP)$$

Therefore, based on the proof by contradiction we prove Lemma 1, PP can exist if a transition exists once and only once in PP. □

Lemma 2. If ($\exists$PP) Then ($\forall t \in T, t \in PP$)

Proof. Based on lemma 1, if a transition exists in PP, then it can only exist once and based on the loop step 2 in our algorithm, the algorithms will generate PP and

terminate once T.num transitions are added to PP as shown on line 19, otherwise the execution terminates with an error report without a generation of PP and:

$$\text{If } (\exists PP) \text{Then } \{(\ \forall i, j \in N \ \text{ and } \ i, j < T.num \text{ and } i \neq j, \quad t_i \neq t_j) \text{ And } (|PP| = T.num)\}$$

Therefore, by direct proof, we prove Lemma 2, PP can exist if all transitions in T exist in PP. □

Lemma 3. $\forall i \in N \text{ and } i \leq n, \ \{\forall t_i \in T_i, t_i \ enabled / \forall t_{i-1} \in T_{i-1}, t_{i-1} \ fired\}$

Proof. We prove this Lemma by mathematical induction.

Basis step: for i=0, loop step 1 clears *A* from all input places with initial markings and adds all transitions to *PP* having inputs with only initial markings (from XCD nodes or users). Since all of the transitions in CL_0 have only input places with initial markings, therefore:

$$\forall t_0 \in T_0, t_0 \ enabled$$

Inductive step: consider $k<n$, we assume that $\forall t_k \in T_k, t_k \ enabled / \forall t_{k-1} \in T_{k-1}, t_{k-1} \ fired$.

Since $\forall t_k \in T_k, t_k \ enabled$ therefore all t_k in T_k are ready to fire. Based on loop step 2, once all t_k fires, all of their output places are cleared from A (line 24). Based on the hypothesis, a place can either have one token from an initial marking or from a fired transition, and since all transitions with initial markings have already fired in the basis step and their places were cleared from *A*, therefore the places left in *A* can obtain a token only from fired transitions. Once all t_k fire, the input places of t_{k+1} which are the

output places of t_k are cleared (line 27) and thus all t_{k+1} are enabled having no input places left in *A*. Thus we conclude by induction that:

$$\forall t_{k+1} \in T_{k+1}, t_{k+1}\ enabled / \forall t_k \in T_k, t_k\ fired \square$$

Now that we have presented our algorithm for discovering and generating concurrent processing sequences corresponding to the resulting composition, we give a detailed illustration showing the results of each executed iteration.

Illustration

Consider the CP-Net shown in Figure 11. Table V represents its Incidence Matrix.

Tab.V: Incidence Matrix of CPN_1

	P/T	**T_0**	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**T_6**
***→**	**P_0**	-1						
	P_1	1	-1					
	P_2		1	-1				
	P_3		1				-1	
→*	**P_4**			1				
***→**	**P_5**				-1			
	P_6				1	-1		
	P_7					1	-1	
	P_8						-1	1
→*	**P_9**						1	
***→**	**P_{10}**							-1

The first iteration terminates after executing the first loop step, where the transitions attached to source places "*→" (XCD-nodes or User places) which must be fired first are generated from Table VI and inserted in CL_0 as shown in Table VII.

Tab.VI: Incidence Matrix after iteration 1

	P/T	T_0	T_1	T_2	T_3	T_4	T_5	T_6
*→	P_0							
	P_1	1	-1					
	P_2		1	-1				
	P_3		1				-1	
→*	P_4			1				
*→	P_5							
	P_6				1	-1		
	P_7					1	-1	
	P_8						-1	1
→*	P_9						1	
*→	P_{10}							

Tab.VII: PP matrix after iteration 1

CL/T	T_0	T_1	T_2	T_3	T_4	T_5	T_6
CL_0	0			1			2

The second iteration is executed in the second loop step for a CL_i=CL_1. The execution terminates after i gets incremented by 1.

Table IX shows the added transitions which must be executed in CL_1.

Tab.VIII: Incidence Matrix after iteration 2

	P/T	T_0	T_1	T_2	T_3	T_4	T_5	T_6
*→	P_0							
	P_1							
	P_2		1	-1				
	P_3		1				-1	
→*	P_4			1				
*→	P_5							
	P_6							
	P_7					1	-1	
	P_8							

Tab.IX: PP matrix after iteration 2

CL/T	T_0	T_1	T_2	T_3	T_4	T_5	T_6
CL_0	0			1			2
CL_1		4			5		

→*	P_9						1	
*→	P_{10}							

The third iteration is executed for i=2. The results are shown in Table X and XI.

Tab.X: Incidence Matrix after iteration 3

	P/T	T_0	T_1	T_2	T_3	T_4	T_5	T_6
*→	P_0							
	P_1							
	P_2							
	P_3							
→*	P_4			1				
*→	P_5							
	P_6							
	P_7							
	P_8							
→*	P_9						1	
*→	P_{10}							

Tab.XI: PP matrix after iteration 3

CL/T	T_0	T_1	T_2	T_3	T_4	T_5	T_6
CL_0	0			1			2
CL_1		3			4		
CL_2			5			6	

Then, the algorithm checks that all the transitions are available once and only once in the PP matrix and ends the execution. Therefore, we conclude that in this case, 3 iterations where required in order to generate the PP matrix. As it is shown in Table XI, the PP matrix contains 3 CL which must be executed from CL_0 to CL_2 sequentially. All transitions available in the same CL can be executed in parallel. As for a serial execution, we can see in the resulting PP matrix that a unique number is associated to each transition which specifies its serial execution order.

Prototype and Experiments

In order to validate our framework and test the algorithm, we implement a prototype called XA2C. It is based on the XCDL core grammar allowing us to compose/draw XML oriented manipulation operations based on functions existing in the system libraries (DLLs, Jars or Web Services). The prototype, illustrated in Figure 12, was developed in Visual Basic Dot Net (VB.Net).

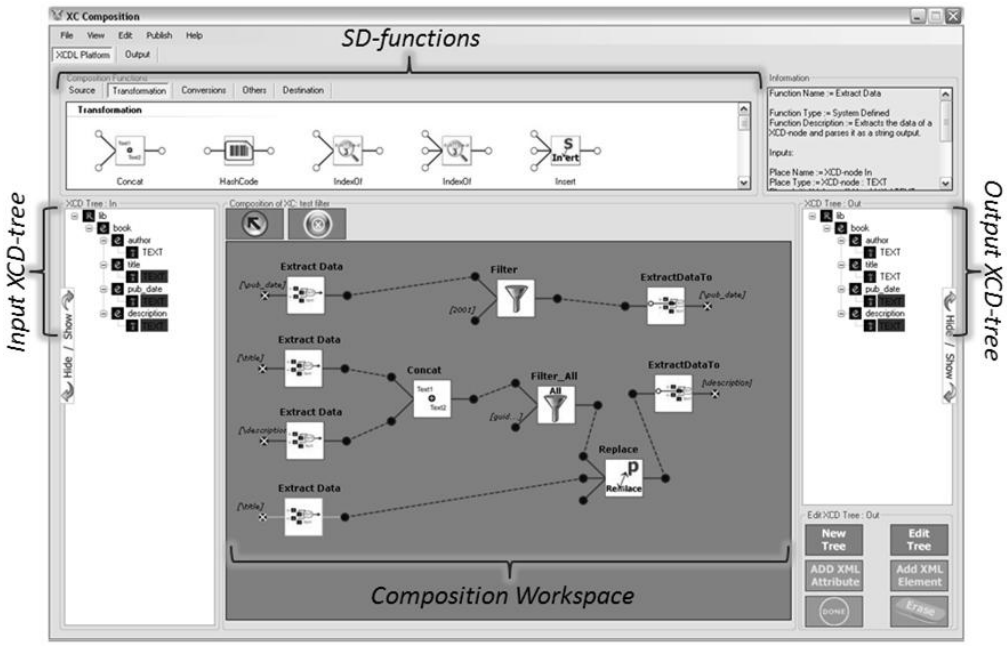


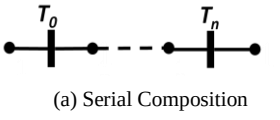
Fig.12: XCDL Platform

The architecture of the prototype is shown in Figure 5. The primary tests we run on the prototype were divided into 3 categories: (i) performance, (ii) user satisfaction, and (i) relevance. In this section, and due to the lack of space we will focus on evaluating the performance and in particular, the process sequence generator algorithm which was implemented in the third module, in the Process Sequence Generator component.

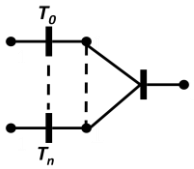
We tested our algorithm for several compositions on an Intel Xeon 2.66GHz with 1Gbyte of Ram memory. We discuss here 4 different cases: serial (cf. Figure 13.a), concurrent (cf. Figure 13.b) and 2 cases of combined and independent compositions: serial and concurrent (cf. Figure 13.c and 13.d). The combined cases can contain several serial composition with concurrent compositions such as, in case a and b, but the combinations are independent and do not share any data, in other words they have no mappings between each other.

In all 4 cases, the functions were dragged and dropped arbitrarily. The runtime execution monitored by the tests regarding cases a, b, c and d are shown respectively in the graphs a, b, c and d in Figure 14.

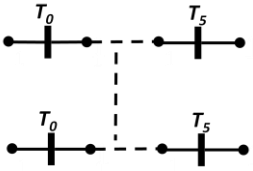
As we can see in all 4 graphs, the runtime execution growth remains constant to a certain point then starts growing in almost a linear form. Therefore, we elaborate the following 4 equations shown in Table XII.



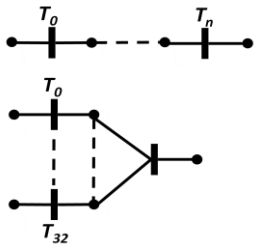
(a) Serial Composition



(b) Concurrent Composition

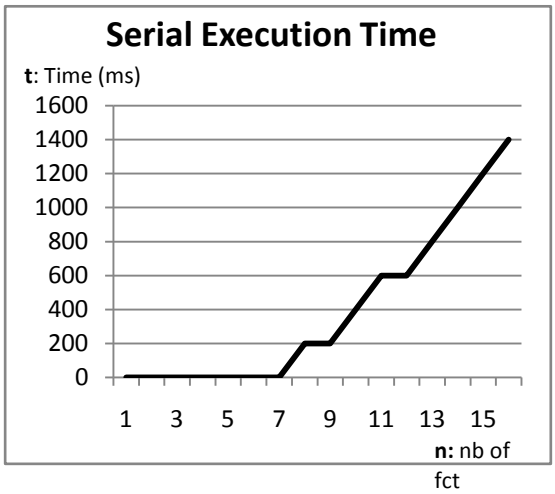


(c) Combined Composition

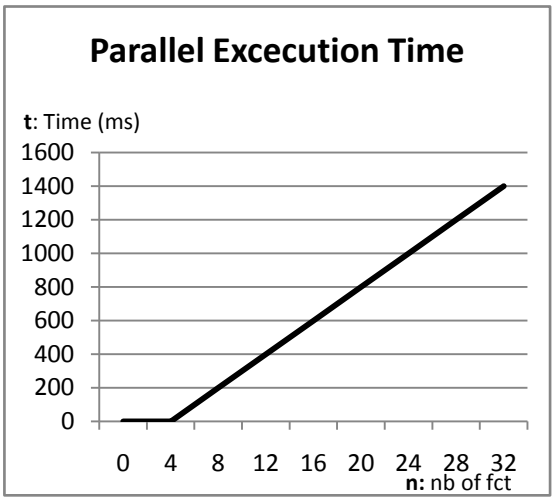


(d) Combined Composition

Fig.13: Different Composition Scenarios



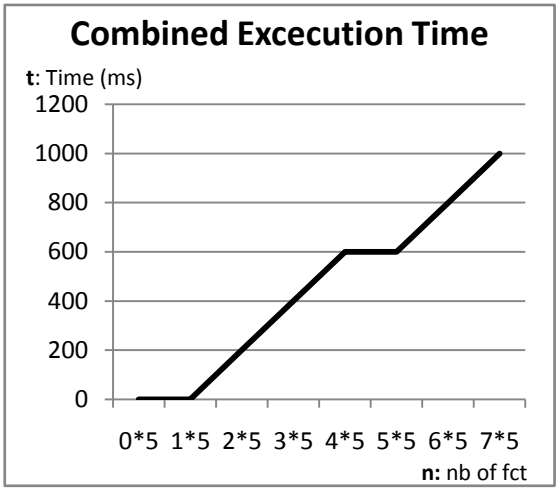
(a) Graph of Case a



(b) Graph of Case b



(c) Graph of Case c



(d) Graph of Case d

Fig.14: Runtime Execution of the Algorithm

Tab.XII: Runtime Equations of cases a, b, c and d

Cases	Runtime Growth Equation
Case a	t = 200n – 7*(200)
Case b	t = 50n – (200)
Case c	t = 320n – (200)
Case d	t = 300n – 6*(200)

Based on all 4 equations, we elaborated that the algorithm has a constant execution period, in the case of the Xeon processor it was 200ms. The execution runtime of concurrent cases is half the execution runtime of serial cases. In combined and independent compositions, we notice that the execution runtime of the algorithm is dependent of the runtime of the maximum independent concurrent composition which sets the minimum runtime of the overall execution.

Conclusion and Future Works

In this paper, we discussed the issues regarding XML manipulations by both expert and non expert users. We were mainly interested in XML-oriented visual languages and XML adaptation/alteration techniques. In terms of visual languages, we identified 2 main approaches, Mashups and XML oriented visual querying languages. On one hand, Mashups are not XML specific and have no been formally defined yet, and on the other hand, XML-oriented languages are limited to data extraction instead of manipulation (insertion, deletion, modification, etc.), are mainly based on existing languages and have limited expressiveness. As for the Alteration/Adaptation techniques, they are intended for experts only, and the current techniques are separate from each other and not necessarily generic to all XML data. To solve these issues, we introduced the XA2C framework XML oriented visual manipulation based on functional compositions where the adaptation/alteration techniques are used as existing functions which can be initiated either from offline libraries (DLL or JAR Files) or online libraries (Web Services). This paper gave a brief introduction to the XCDL language which was defined based on CP-Nets and intended to be used for visual functional compositions. The paper also presented the algorithm we developed

in order to discover and generate serial and concurrent processing sequences resulting from the compositions created by the XCDL language. The algorithm was implemented and tested in a prototype developed in VB.Net which allows users to create composed operations for XML textual values mainly. The main track, in future works, on one hand, relies on optimizing the algorithm to deal with independent compositions in more efficient manner in terms of timing and error handling. On the other hand, it relies on extending the XCDL language to grow beyond functional compositions by adding conditional and loop operators along with user composed functions which can be reused in different compositions.

References

ADELBERG, B. 1998. NoDoSE-a tool for semi-automatically extracting structured and semistructured data from text documents. *SIGMOD Rec.*, 27**,** 283-294.

ALTINEL, M. & FRANKLIN, M. J. 2000. Efficient Filtering of XML Documents for Selective Dissemination of Information. *Proceedings of the 26th International Conference on Very Large Data Bases.* Morgan Kaufmann Publishers Inc.

BRAGA, D., CAMPI, A. & CERI, S. 2005. XQBE (XQuery By Example): a visual interface to the standard XML query language. *ACM Trans. Database Syst.*, 30**,** 398-443.

BYUN, C., LEE, K. & PARK, S. 2007. A Keyword-Based Filtering Technique of Document-Centric XML using NFA Representation. *International Journal of Applied Mathematics Computer Science***,** 136–143.

CANDAN, K. S., HSIUNG, W.-P., CHEN, S., TATEMURA, J. & AGRAWAL, D. 2006. AFilter: adaptable XML filtering with prefix-caching suffix-clustering. *Proceedings of the 32nd international conference on Very large data bases.* Seoul, Korea: VLDB Endowment.

CERI, S., COMAI, S., DAMIANI, E., FRATERNALI, P. & TANCA, L. 2000. Complex queries in XML-GL. *Proceedings of the 2000 ACM symposium on Applied computing - Volume 2.* Como, Italy: ACM.

CHANG, C.-H. & LUI, S.-C. 2001. IEPAD: information extraction based on pattern discovery. *Proceedings of the 10th international conference on World Wide Web.* Hong Kong, Hong Kong: ACM.

CRESCENZI, V., MECCA, G. & MERIALDO, P. 2002. Automatic Web Information Extraction in the ROADRUNNER System. *Revised Papers from the HUMACS, DASWIS, ECOMO, and DAMA on ER 2001 Workshops.* Springer-Verlag.

DIAO, Y., ALTINEL, M., FRANKLIN, M. J., ZHANG, H. & FISCHER, P. 2003. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Trans. Database Syst.*, 28**,** 467-516.

ENNALS, R. J. & GAROFALAKIS, M. N. 2007. MashMaker: mashups for the masses. *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. Beijing, China: ACM.

ERWIG, M. 2000. A Visual Language for XML. *Visual Languages, IEEE Symposium on*, 0**,** 47.

GOLIN, E. J. & REISS, S. P. 1990. The specification of visual language syntax. *J. Vis. Lang. Comput.*, 1**,** 141-157.

HWANG, G.-H. & CHANG, T.-K. 2004. An operational model and language support for securing XML documents. *Computers & Security*, 23**,** 498 - 529.

JENSEN, K. 1994. An Introduction to the Theoretical Aspects of Coloured Petri Nets. *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium.* Springer-Verlag.

LEMLOUMA, T. & LAYAÏDA, N. Year. SMIL Content Adaptation for Embedded Devices. *In:* IN SMIL EUROPE 2003 CONFERENCE, 2003. 12--14.

LORENZO, G. D., HACID, H., PAIK, H.-Y. & BENATALLAH, B. 2009. Data integration in mashups. *SIGMOD Rec.*, 38**,** 59-66.

LOTON, T. 2008. *Introduction to Microsoft Popfly, No Programming Required*, Lotontech Limited.

LUO, B., LEE, D., LEE, W.-C. & LIU, P. 2004. QFilter: fine-grained run-time XML access control via NFA-based query rewriting. *Proceedings of the thirteenth ACM international conference on Information and knowledge management.* Washington, D.C., USA: ACM.

MURATA, T. 1989. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE.*

PELLAN, B. & CONCOLATO, C. 2008. Adaptation of scalable multimedia documents. *Proceeding of the eighth ACM symposium on Document engineering.* Sao Paulo, Brazil: ACM.

PIETRIGA, E., VION-DURY, J.-Y. & QUINT, V. 2001. VXT: a visual approach to XML transformations. *Proceedings of the 2001 ACM Symposium on Document engineering.* Atlanta, Georgia, USA: ACM.

SIMMEN, D. E., ALTINEL, M., MARKL, V., PADMANABHAN, S. & SINGH, A. 2008. Damia: data mashups for intranet applications. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data.* Vancouver, Canada: ACM.

TEKLI, G., CHBEIR, R. & FAYOLLE, J. 2010a. XA2C Framework for XML Alteration/Adaptation. *In:* SHIN, S. Y., GANTENBEIN, R., KUO, T.-W. & HONG, J. (eds.) *Reliable and Autonomous Computational Science.* Springer Basel.

TEKLI, G., CHBEIR, R. & FAYOLLE, J. Year. XCDL: an XML-Oriented Visual Composition Definition Language. *In:* The 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS2010), 2010b.

TEKLI, G., FAYOLLE, J. & CHBEIR, R. 2010c. Towards an XML Adaptation/Alteration Control Framework. *Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services.* IEEE Computer Society.

W3C 1999. Extensible Stylesheet Language Transformations -XSLT 1.0.

Short Biographies

Gilbert Tekli received his Masters of Engineering in Telecommunications and Computers from the Antonine University-Lebanon in 2007. He is currently completing his Phd thesis in Telecom St Etienne-France, co-directed by the LE2I laboratory in the Computer Science Department, University of Bourguogne-France. His work and research interest fall mainly in the areas of XML manipulation/control, dataflow, visual languages, formal languages, EDRM and access control models.

Gilbert Tekli holds a teaching position in Telecom St Etienne where he mainly teaches Security (e.g., EDRM and Information System Protection) and programming (e.g., Algorithm and Java) courses.

Richard Chbeir received his PhD in Computer Science from the University of INSA-FRANCE in 2001and then his Habilitation degree in 2010 from the University of Bourgogne where he is currently an Associate Professor in the Computer Science Department in Dijon-France. His research interests are in the areas of multimedia information retrieval, XML and RSS Similarity, access control models, multimedia document annotation.

Richard Chbeir has published in international journals, books, and conferences, and has served on the program committees of several international conferences He is currently the Chair of the French Chapter ACM SIGAPP and the Vice-Chair of ACM SIGAPP. Richard Chbeir teaches Databases and Multimedia Data Retrieval in the Computer Science Department of the "IUT de Dijon" and "UFR Science et Technique" of Bourgogne University in Dijon-France.

Jacques Fayolle is co director of Télécom Saint-Etienne (http://www.telecom-st-etienne.fr), engineering school in information technologies. He drives the local research activities on interoperability in information systems and the adaptation to the context of applications. He received his PhD in computer science in 1996 and the French habilitation for research management in 2007. Jacques is the author or co author of more than 70 scientific publications in international journal or symposium. His last publication (in co author with C. Gravier) deals with the use of semantic tools for remote lab session and how improves the quality of learning (IEEE Intelligent Systems).