Modeling and Verifying Active XML Artifacts

Serge Abiteboul, Luc Segoufin, Victor Vianu

Modeling and Verifying Active XML Artifacts *

Serge Abiteboul
INRIA Saclay & LSV - ENS Cachan, France

Luc Segoufin
INRIA & LSV - ENS Cachan, France

Victor Vianu
U.C. San Diego, USA

1 Introduction

Shared evolving data is central to an increasing range of human activities. In response to the need for computerized support of such activities, the notion of *business artifact* has been proposed at IBM as a model of such evolving data [1]. The model captures both the flow of control (workflow) of the application and the evolution of the relevant data (data cycle); see [2] for a brief survey. In the same spirit, we propose a new artifact model building upon Active XML (AXML for short), an extension of XML with embedded service calls [3]. The services are hosted by autonomous peers that evolve and interact by exchanging XML data. We claim that this can provide the foundation for an appealing artifact model, combining the advantages of semistructured data and of the Web service paradigm. With the model in place, we consider the verification of data-intensive applications, which is particularly critical for such systems due to their vulnerability to costly bugs. Despite the expressiveness of the model, we show that verification remains possible under reasonable restrictions.

Workflow and database systems are two essential software components that often have difficulties interoperating. Data-centric workflow systems are meant to integrate the control aspect of workflows with the underlying data. They allow managing data evolution by tasks with complex sequencing constraints as encountered for instance in scientific workflow systems, information manufacturing systems, e-government, e-business or healthcare systems. One can distinguish two main approaches for combining the database and workflow components. One consists in starting from a workflow approach, enriching it with data, e.g., by explicitly introducing state variables and specifying how they may evolve. The second emphasizes data placed at the center of the specification, but enriches it with means of controlling how it evolves. There is no fundamental separation between these two kinds of approaches but more a bias coming from where the emphasis is placed. However, when an emphasis is placed on the data (as we do here), one tends to prefer declarative specifications based on constraints on the evolution rather than control-based specifications.

We follow here a data-centric workflow approach where both data and tasks, but also the "actors" (humans, processes, systems) are captured by AXML *artifacts* [4]. The basis of this work is thus the Active XML model. AXML documents [3, 5] are XML documents with embedded function calls realized as Web service calls. Observe that the central notion is a document, so data, but that the model also involves computation, i.e., Web services. A main issue in the AXML technology is "when is a Web service call" evaluated. In query processing, a call may be activated because its result may impact the result of a query; this is in the spirit of recursive query processing. A call may also be activated bacause some event occurred, as in active databases. In the present

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Work supported by the Webdam Grant of the European Research Councild (FP7); n. 226513. The work of the first two authors is also supported in part by the EC FoX Grant. The work of the last is supported in part by the National Science Foundation under grant number IIS-0916515.

work, we want activation to be guided by the logic of the application, e.g. by some workflow constraints. In particular, we want to be able to specify some particular sequencings of the Web service calls inside a document.

An example of AXML documents is shown in Figure 3. Function *warehouseOrder* is used to obtain the parts from a warehouse. Function *deliveryOrder* is used to start the delivery process. It is likely that the system will delay the activation of this second function until after the computer has been built. On the other hand, the parts should be obtained from the warehouse before the construction starts.

The calls in an AXML document may be activated from inside (the artifact as client) and then receive answers in push or pull mode. Calls may also be activated from outside (the artifact as server). Rules are used to specify the logic of functions declaratively [6]. We use such documents to represent artifacts. In the spirit of [7, 1], an AXML document represents a process that evolves in time. A function call may be seen as a request to carry out a subtask whose result may lead to a change of state in the document.

An Active XML system specifies a set of interacting AXML documents. In such a system, there is an important distinction between internal and external services. An internal service is a service that is completely specified within the system whereas an external one captures interactions with other services or with users. One important goal is to statically analyze the behavior of such systems, which is especially challenging because the presence of data induces infinitely many states. We illustrate this aspect by mentioning some work on the verification of restricted centralized AXML systems [6]. These results can be easily transferred to distributed systems of AXML artifacts.

In this paper, we briefly present the AXML artifact model [4] (Section 2). We also mention some work on data-centric verification from [6] (Section 3). The last section provides brief conclusions.

2 The AXML Artifact model

Artifacts present several facets that, in our opinion, should be captured by an artifact model. An artifact is an *object* with a universal identity (e.g., URI). Its *state* is self-describing (e.g., XML data) so that it may be easily transmitted or archived. An artifact may host other artifacts as components, yielding a hierarchy of artifacts. At the physical level, each artifact at the root of the hierarchy is hosted by a peer. During its life cycle, an artifact is created, evolves in time, migrates among hosts, may hibernate and be reactivated, or dies according to a logic that is specified declaratively. Its *evolution* may be constrained to obey some laws, e.g. a *workflow*. An artifact *interacts* with the rest of the world via function calls (e.g., Web services) both as a server and a client. An artifact provides for communications, storage and processing for the artifacts it hosts. As in scientific workflows, an artifact has a *history* including time and provenance information that may be recorded and queried. These requirements have been in part motivated by [8].

To illustrate, consider a simplified view of the Dell manufacturing system [9] (Figure 1). When a new Web order arrives (1), a new *webOrder* artifact is created and creates a subartifact that is sent to a credit service (2). Once credit has been approved, the subartifact returns to the *webOrder* but now its state contains all the credit data. A plant is then selected and the artifact moves to that plant (3). It initiates a new subartifact for gathering parts, that is sent to a warehouse and another local artifact for communications with the customer (4). Once the product has been built, the artifact is sent to a delivery service (5). Finally, once the Web order has been completed, the artifact moves to an archive where it is stored as a text-based XML serialization that includes all the information it has gathered during its life cycle (6). (Subartifacts may also be archived separately.) The Dell example can be naturally modeled in AXML. See Figure 3 where the tree is represented using an XML syntax (a text-based serialization of the tree). The figure shows part of a *webOrder* artifact immediately after it enters the plant. The *creditApproval* element denotes a subartifact (the one that has been processed by the bank). The functions *?warehouseOrder* and $?comm$ will be activated next in order to create the *warehouseOrder* and *communication* subartifacts that will then work concurrently (and somewhat autonomously).

More broadly, the use of AXML as a basis for an artifact model is motivated by the fact that it can be easily

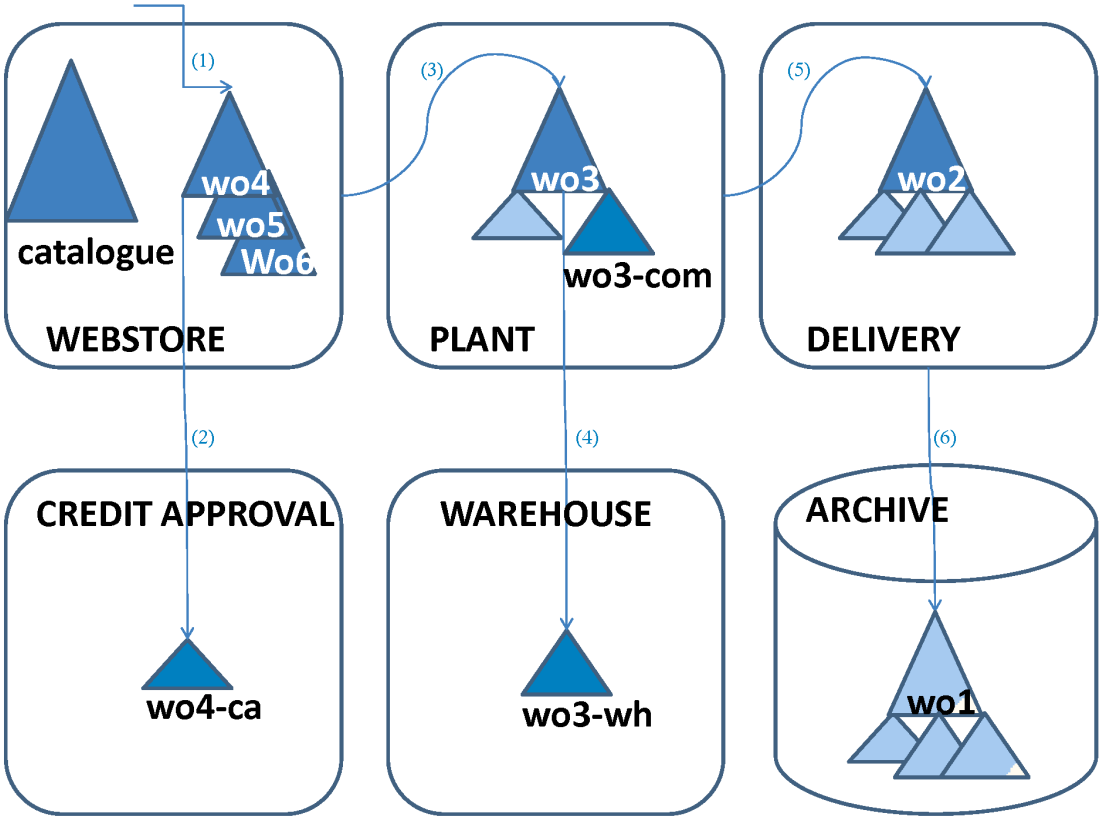

Figure 1: Artifacts in the Dell application

adapted to support all the requirements identified above. In particular, AXML naturally captures the distribution and autonomy of artifacts and provides reliable synchronous or asynchronous communication, allowing an artifact to send a message to another artifact just by knowing its ID. Also, because of its nested structure, AXML naturally supports hierarchies of artifacts. Two functionalities have to be added to AXML in order to fully support the above requirements. First, since we want artifacts to move from place to place in the system, we need an identification mechanism serving as a URI for artifacts. We also augment the rule-based workflow specification provided by AXML with workflows specified in a transition-based BPEL style that is more familiar to application designers.

The core of an application is a *schema* specifying a set of of peers and a set of classes (e.g., webOrder, financialService). The definition of a class provides typing of the data (document types), dynamic constraints on artifacts evolution (their workflows) and the interface of functions that the artifacts in this class export. From an implementation viewpoint, a peer provides storage, communications and computing resources for the artifacts it hosts. Artifacts are allowed to exchange data with other peers or to move to other peers. AXML data (e.g., in function arguments and results) is sent as strings and reconstructed at the receiving peers.

The semantics of functions is specified by rules. The declarative semantics facilitates reasoning about the runs of such systems and performing optimization. Function call activation is controlled by *call guards* that are specified by *Boolean combinations of tree-patterns* over the documents. Observe that the guards impose

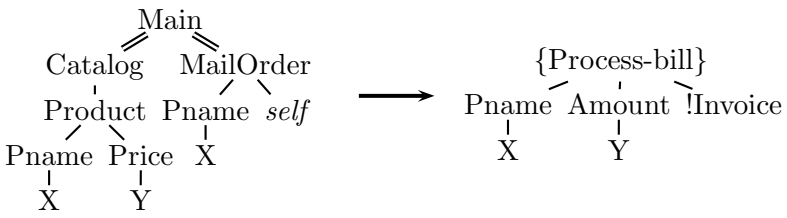

Figure 2: A GAXML query

```
< plant artID="plant02" >
...
< webOrder artID="wo3" >
    < client >
      < name > Sue Leroux < /name >
      < address > ... < /address >
    < /client >
    < order > ... < /order >
    < order > ... < /order >
  < creditApproval artID="wo3-ca" >
   ...
  < /creditApproval >
  < fun funID="?warehouseOrder"/ >
  < fun funID="?deliveryOrder"/ >
  < fun funID="?comm"/ >
< /webOrder >
...
< /plant >
```

Figure 3: An AXML artifact

constraints on the evolution of documents in the style of condition-action rules. This may be seen as specifying *workflow* constraints on the runs of the system. Alternatively, one might prefer a more standard workflow approach in the style BPEL. The workflow is then specified by defining *stages* in the evolution of the artifact and admissible transitions between them. We are currently working on a comparison of the two styles of workflow specifications.

The notions of *task*, *service*, *state*, *stage*, and *activity*, that are essential in the artifact context, can all be formally captured in the AXML artifact model. The notion of *activity*, often arising in functional decompositions of business processes, may be seen as a view over a system of artifacts. The notions of *time* and *provenance* that are central to scientific workflows can be captured as well, because this information can be recorded and maintained in XML documents.

Related work Although the notion of artifact has been recently articulated by [1], similar ideas of data centric workflows have been around, e.g., in AXML [3], in the Vortex system [10] or scientific workflows [11]. The models that are considered are often restricted, e.g., [12, 8]. For instance, a single artifact is usually considered, vs. a system of artifacts in the present paper. Also, these models are often based on the relational model so have difficulties with collections of artifacts or nested tasks/artifacts. Formal models for data-centric workflows have been considered in [13] (that focuses on verification) and [14] (that discusses the synthesis of artifacts).

3 Verification

The need for reasoning about artifact systems arises in many contexts and is particularly challenging because of the presence of data. We briefly summarize results obtained in [6] on static analysis of AXML systems, in particular on automatic verification of temporal properties of their runs.

Classical automatic verification techniques operate on finite-state abstractions that ignore the critical semantics associated with data in such applications. The need to take into account data semantics has spurred interest in studying static analysis tasks in which data is explicitly present. We have started an investigation of the automatic verification of Active XML systems. We consider properties expressed in Tree-LTL, an extension of LTL

where propositions are interpreted as tree patterns. For instance, one may want to verify whether some static property (e.g., all ordered products are available) and some dynamic property (e.g. an order is never delivered before payment is received) always hold. Tree-LTL allows to express a rich class of such properties. An example of Tree-LTL formula can be found in Figure 4.

We have identified a significant fragment of Active XML, called non-recursive Guarded AXML for which the verification of Tree-LTL properties is decidable. This fragment is expressive enough to describe meaningful applications. We use also it as a convenient formal vehicle for studying decidability and complexity boundaries for verification in AXML in general. We briefly describe next Guarded AXML (GAXML for short) and its non-recursive fragment.

Every product for which a correct amount has been paid is eventually delivered (note that the variable Z is implicitly existentially quantified in the left pattern):

$$\forall X \forall Y [\mathbf{G}(\quad \rightarrow \mathbf{F}(\quad))]$$

Main — Catalog — Product — Pname — X; Price — Z; MailOrder — Paid — Pname — X; Amount — Z; Order-Id — Y

Main — MailOrder — Pname — X; Order-Id — Y; Delivered

Figure 4: A Tree-LTL formula

In GAXML, document trees are unordered. With ordered trees, verification quickly becomes undecidable. Finally, the most novel feature of the model in the AXML context is a *guard* mechanism for controlling the initiation and completion of subtasks (formally function calls). Guards are Boolean combinations of tree patterns. They facilitate specifying applications driven by complex workflows and, more generally, they provide a very useful programming paradigm for active documents.

We obtain decidability by disallowing recursion in GAXML systems, which leads to a static bound on the total number of function calls in runs. We prove that for such non-recursive GAXML, satisfaction of Tree-LTL formulas is CO-2NEXPTIME-complete. We also consider various relaxations of the non-recursiveness restriction and show that they each lead to undecidability. This establishes a fairly tight boundary of decidability of satisfaction of Tree-LTL properties by GAXML systems.

Related work Most of the previous work on static analysis on XML (with data values) deals with documents that do not evolve in time (static constraints). This motivated studies of automata and logics on strings and trees over infinite alphabets, see [15] for a survey. Previous work on AXML also considered the evolution of documents. For instance, this is considered in [16] for a monotone AXML language, *positive* AXML. The setting is very different from ours, as their systems are monotone but possible recursive. In contrast, we consider verification for nonmonotone systems. Static analysis is also studied in [17] using a model based on tree rewriting. Verification of temporal properties of Web services has mostly been considered using models abstracting away data values (see [18] for a survey). Verification of data-aware Web services was studied in [19, 20], and a verifier implemented [21]. While this is related in spirit to the present work, the technical differences stemming from the AXML setting render the two investigations incomparable.

4 Conclusion

We briefly mention some remaining issues related to the work presented here. A most interesting direction of research is to enrich beyond non-recursive GAXML the class of AXML artifact systems that can be verified. For example, one would also like to be able to reason about time (this is complicated for several reasons, including the absence of a global clock). When full verification cannot be performed, abstraction may be useful. Recent work considers a related approach based on *interfaces* in the context of AXML [22]. Besides verification, a main issue for such systems is monitoring. A P2P monitoring system for AXML is studied in [23, 24]. Finally, largely unexplored in this context are access control mechanisms, allowing different actors to keep control over their own data without imposing unacceptable constraints on the system.

References

[1] A. Nigam and N. Caswell, "Business artifacts: An approach to operational specification," *IBM Systems Journal*, vol. 42, no. 3, pp. 428–445, 2003.

[2] R. Hull, "Artifact-centric business process models: Brief survey of research results and challenges," in *OTM*, 2008.

[3] S. Abiteboul, O. Benjelloun, and T. Milo, "The Active XML project: an overview," *VLDB J.*, 2008.

[4] S. Abiteboul, P. Bourhis, A. Galland, and B. Marinoiu, "The axml artifact model," in *16th International Symposium on Temporal Representation and Reasoning (TIME-2009)*, 2009.

[5] Active XML, "http://activexml.net."

[6] S. Abiteboul, L. Segoufin, and V. Vianu, "Static analysis of Active XML systems," in *PODS*, 2008, pp. 221–230, full paper in *ACM TODS* 34:3, 2009.

[7] R. Khalaf, A. Keller, and F. Leymann, "Business processes for web services: Principles and applications," *IBM Systems Journal*, no. 45:2, 2006.

[8] K. Bhattacharya, R. Hull, and J. Su, "A data-centric design methodology for business processes," in *Handbook of Research on Business Process Management*, J. Cardoso and W. van der Aalst, Eds. N.A., 2009.

[9] R. Kapuscinski, R. Q. Zhang, P. Carbonneau, R. Moore, and B. Reeves., "Inventory decisions in Dell's supply chain," *Interfaces*, vol. 34, no. 3, pp. 191–205, 2004.

[10] G. Dong, R. Hull, B. Kumar, J. Su, and G. Zhou, "A framework for optimizing distributed workflow executions," in *DBPL 1999*, 2000.

[11] S. Davidson and J. Freire, "Provenance and scientific workflows: Challenges and opportunities," in *ACM SIGMOD Int. Conf. on Management of Data*, 2008.

[12] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su, "Towards formal analysis of artifactcentric business process models," in *Int. Conf. on Business Process Management*, 2007.

[13] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu, "Automatic verification of data-centric business processes," in *ICDT*, 2009.

[14] C. Fritz, R. Hull, and J. Su, "Automatic construction of simple artifact-based business processes," in *ICDT*, 2009.

[15] L. Segoufin, "Static analysis of xml processing with data values," *Sigmod Record*, no. 36:1, 2007.

[16] S. Abiteboul, O. Benjelloun, and T. Milo, "Positive Active XML," in *ACM PODS*, 2004, pp. 35–45.

[17] B. Genest, A. Muscholl, O. Serre, and M. Zeitoun, "Tree pattern rewriting systems," in *ATVA, LNCS 5311*, 2008.

[18] R. Hull, M. Benedikt, V. Christophides, and J. Su, "E-services: a look behind the curtain," in Proc. *ACM PODS*, 2003.

[19] A. Deutsch, L. Sui, and V. Vianu, "Specification and verification of data-driven web applications," *J. Comput. Syst. Sci.*, vol. 73(3), 2007.

[20] A. Deutsch, L. Sui, V. Vianu, and D. Zhou, "Verification of communicating data-driven web services," in Proc. *ACM PODS*, 2006.

[21] D.-D. W. A. A Verifier for Interactive, "A. deutsch and m. marcus and l. sui and v. vianu and d. zhou," in Proc. *ACM SIGMOD*, 2005.

[22] A. Benveniste and L. Helouet, "Interface for Active XML," private communication.

[23] S. Abiteboul and B. Marinoiu, "Distributed Monitoring of Peer to Peer Systems," in *Workshop On Web Information And Data Management*, 2007, pp. 41–48.

[24] S. Abiteboul, P. Bourhis, and B. Marinoiu, "Satisfiability and Relevance for Queries over Active Documents," in *PODS 2009*, 2009.