The IT University of Copenhagen

Distributed Reactive XML

an XML-centric coordination middleware

Thomas Hildebrandt
Henning Niss
Martin Olsen
Jacob W. Winther

IT University Technical Report Series **TR-2005-62**

ISSN 1600–6100 **March 2005**

ISSN 1600–6100

ISBN 87-7949-091-3

Copies may be obtained by contacting:

IT University of Copenhagen
Glentevej 67
DK-2400 Copenhagen NV
Denmark

Telephone: +45 38 16 88 88
Telefax: +45 38 16 88 99
Web `www.itu.dk`

Distributed Reactive XML

an XML-centric coordination middleware

Thomas Hildebrandt, Henning Niss, Martin Olsen, and Jacob Winther*

IT University of Copenhagen {hilde,hniss,mol,jww}@itu.dk

Abstract. XML-centric models of computation have been proposed as an answer to the demand for interoperability, heterogeneity and openness in coordination models. We present a prototype implementation of an open XML-centric coordination middleware called Distributed Reactive XML. The middleware has as theoretical foundation a general *extendable*, *distributed* process calculus inspired by the theory of *Bigraphical Reactive Systems*. The calculus is *extendable* just as XML is extendable, in that its signature and reaction rules are not fixed. It is *distributed* by allowing both the *state* of processes as well as the set of *reaction rules* to be distributed (or partly shared) between different clients. The calculus is implemented by representing process terms as XML documents stored in a value-oriented, peer-to-peer XML Store and reaction rules as XML transformations performed by the clients. The formalism does not require that only process terms are stored—inside process terms one may store application specific data as well. XML Store provides transparent sharing of process terms between all participating peers. Conflicts between concurrent reaction rules are handled by an optimistic concurrency control. The implementation thus provides an open XML-based coordination middleware with a formal foundation that encompasses both the shared data, processes and reaction rules.

1 Introduction

The ubiquity of XML as a format for exchange and processing of semi-structured data has naturally led to research in the interplay between XML and programming languages and models for global ubiquitous computing. It was early on observed that the Mobile Ambient calculus, the seminal calculus for nested mobile computing agents, describes reconfigurations of semi structured data [3]. It was suggested that this relationship could permit transfer of techniques in both directions, e.g. using so-called spatial logics for mobile process calculi to reason about XML data and using semi-structured query languages to search in nested network structures. Following up on these ideas, [4] suggests so-called *XML-centric* models of computation and XML-based middleware for coordination. In XML-centric models of computation the *state* of the computation (or part of the state) consists of XML data. For coordination languages the data is typically stored in a shared (or partly shared) distributed tuple space. The *computation* or *coordination* actions is then expressed in terms of transformations of this XML data.

The use of XML described above to some extent meets the demands for interoperability, heterogeneity and openness in coordination languages and global ubiquitous computing in general [15]. However, the computations are often expressed in general and complex languages such as Java or XSLT. This goes against the hope for obtaining a theory that facilitates analysis of the behaviour of the implemented systems, as advocated in the UK Grand Challenge on Science for Global Ubiquitous Computing. On the other hand, *fixing* a simple set of computation or coordination rules goes against the desire for openness and flexibility.

Recently, *bigraphical reactive systems* [11] have been introduced as a *meta* model for reactive mobile systems with semi structured state. It is a meta model just as XML is a meta

* Authors listed alphabetically. This work was funded in part by the Danish Research Agency (grant no.: 2059-03-0031) and the IT University of Copenhagen (the LaCoMoCo project).

model, in that it allows the definition of domain specific models by specifying the allowed syntax as well as the reaction rules. All bigraph models then benefit from a general theory developed for bigraphical reactive systems, such as e.g. bisimulation proof techniques and spatial logics, as well as the power of being able to translate between different bigraph models.

In the present paper we suggest to utilize the similarities of XML and the theory of bigraphs to implement an open, distributed XML-based coordination middleware with a formal foundation that encompasses both the shared data, processes and reaction rules.

Concretely, we introduce a *distributed, eXtendable Process calculus* (short, the diX-calculus). The diX-calculus is based on a simple extendable calculus of reactive systems, which can be regarded as a notation for XML contexts. It is inspired by the similarities between process calculi for mobility and semi-structured data as observed in [3] and derived from the meta theory of bigraphical reactive systems proposed in [11, 13]. In particular, it is straightforward to provide a semantics for the calculus in bigraphs.[1] The distributed calculus is inspired by the XML-based middleware for coordination investigated in [4], by allowing both the *state* of processes as well as the set of *reaction rules* to be distributed (or partly shared) between different clients.

We present an implementation, called *Distributed Reactive XML*. The processes is stored as XML in a distributed XML store and thereby made accessible to several clients. Each client can perform transformations on the shared XML document according to its own set of reaction rules. An interesting technical contribution is the implementation of concurrency control, dealing with conflicts between concurrent reactions. By analyzing when concurrent reactions are conflicting and storing a complete history of reactions performed, we use this knowledge to implement an optimisitc concurrency control. The reason for using an optimistic approach, as opposed to the lock-based concurrency control for XML documents proposed in [10], is that we use a peer-to-peer network to distribute the XML document. This setting makes it quite complicated to implement a locking mechanism, since we need to ensure that all peers agree on the locks. With the implemented optimistic concurrency control, we only need to ensure that peers agree on the newest version of the document. We implement this optimistic concurrency control using a so called *value-oriented, peer-to-peer distribute XML storage layer* implemented at ITU and DIKU called *XML Store* [2, 9, 14]. In a value-oriented XML Store data is never updated. Instead new values are constructed, reusing old values where possible. This allows a cheap storage of the complete history of updates which are used for detecting conflicts. This history can also be used for backtracking if conflicts are detected or as more general tool for debugging.

Finally, it is worth noting that the formalism does not require that only process terms are stored—inside process terms one may store application specific XML-data as well. The implementation thus provides a simple, open XML-based coordination middleware with a formal foundation that encompasses both shared XML data, processes and reaction rules.

Structure of the paper: In Sec. 2 we present the distributed, eXtendable process calculus (diX). In Sec. 3 we introduce the value-oriented XML Store, and describe Distributed Reactive XML, the prototype implementation of the diX-calculus based on XML Store. In particular we describe how we implement concurrency control. Throughout the paper we use an example of a location-based service. We end in Sec. 4 with pointers to related and future work.

2 A Distributed Extendable Process Calculus

In this section we present a simple distributed eXtendable Process calculus, short the *diX-calculus*, inspired by the similarities between process calculi for mobility and semi-

[1] A bigraph semantics and the possible applications of the general bigraph theory will be addressed in a follow up paper.

structured data as observed in [3] and the meta theory of bigraphical reactive systems proposed in [11, 13].

Notation: We let n, m, i, j range over natural numbers and I and J range over finite sets of natural numbers. We will often confuse a natural number $m \geq 0$ and the set (ordinal) $\{0, 1, \ldots m-1\}$.

2.1 Process expressions

First we define a general notion of signatures that encompasses both the signatures of XML documents and bigraph signatures. The terminology is borrowed from bigraph signatures.

Definition 1. *A* signature *is a tuple* (Σ, N, Att, ar)*, where* Σ *is a set of* controls*,* N *is an infinite set of* names*,* Att *is a set of finite* index sets*, and* $ar : \Sigma \to Att$ *is a function assigning an index set to each control.* □

For the present application, we think of Σ as a set of XML *element names*, N as a set of XML *attribute values*, and Att as finite sets of XML *attribute names*.

Example 1 (Location model). Throughout the paper, we will illustrate the coordination aspects of Distributed Reactive XML with an example from *location-based services*. To make the example manageable it has been simplified a lot; one could easily imagine more complete location-modelling.

The current state of the location example is called the *location state*. A location state is made up of buildings. A building contains a number of floors each with a number of rooms. People can be present in a building, in which case they have to be in some room, or not present in any of the buildings in the location state.

The signature Σ therefore needs to include controls $\mathsf{building}$, floor, room, and person. Some of these controls we adorn with attributes, for example, $name$. The connection between attributes and controls is captured by ar; for example $ar(\mathsf{room}) = \{name\}$. □

We then introduce *process expressions*. It can be seen as a simple process calculus notation for tuples of XML data.

Definition 2. *For a signature* $\Sigma = (\Sigma, N, Att, ar)$ *define the* Σ-process expressions *by the grammar*

$$r ::= r \parallel r \mid p \mid 0 \qquad \text{wide } \textit{processes}$$

$$p ::= \kappa\{i : x_i\}_{i \in ar(\kappa)}.p \mid p \,|\, p \mid 1 \qquad \text{prime } \textit{processes}$$

for $\kappa \in \Sigma$ *and* $x_i \in N$. □

Following [11] we refer to $|$ and $\parallel$ as respectively the *prime* and *wide parallel composition*. We refer to 1 as the *nil process* and to 0 as the *null process*. We assume a structural congruence $\equiv$ on process expressions, making prime parallel composition associative and commutative, wide parallel composition associative, and the nil process 1 and null process 0 respectively the identity for prime and wide parallel composition.

Definition 3. *Structural congruence* $\equiv$ *is the least congruence on process expressions such that*

$$p_1 \,|(p_2 \,|\, p_3) \equiv (p_1 \,|\, p_2) \,|\, p_3 \qquad p \,|\, 1 \equiv p \qquad 1 \,|\, p \equiv p \qquad p \,|\, q \equiv q \,|\, p$$

and

$$r_1 \parallel (r_2 \parallel r_3) \equiv (r_1 \parallel r_2) \parallel r_3 \qquad r \parallel 0 \equiv r \qquad 0 \parallel r \equiv r$$ □

Commutativity of the prime parallel product means that we, as usual in process calculi, consider prime parallel processes unordered. Since we later implement the calculus in terms of ordered XML values we need to carefully treat the values as unordered when processing them. Associativity allows us to leave out parenthesis for prime and wide parallel composition, writing respectively $\mathit{\Pi}_{i\in n}\, p_i$ and $\boldsymbol{\Pi}_{i\in n}\, r_i$ for the n times prime and wide parallel composition and letting $\mathit{\Pi}_{i\in 0}\, p_i = 1$ and $\boldsymbol{\Pi}_{i\in 0}\, r_i = 0$. As usual we will often leave out trailing nil processes, writing $\kappa\{i : x_i\}_{i\in ar(\kappa)}$ for $\kappa\{i : x_i\}_{i\in ar(\kappa)}.1$. We say that the *width* of a wide process expression r is n if $r \equiv \boldsymbol{\Pi}_{i\in n}\, p_i$ for $n \geq 0$, i.e. the process r is the *wide* parallel product of n primes.

Example 2 (CCS and Ambients). We can represent a subset of (finite) CCS as the prime Σ-processes for $\Sigma = \{\mathsf{act}, \mathsf{coact}\}$, Att=$\{\{ch\}\}$ and $ar(\mathsf{act}) = ar(\mathsf{coact}) = \{ch\}$.

We can represent a subset of (finite) Mobile Ambients as the prime Σ-processes for $\Sigma = \{\mathsf{amb}, \mathsf{in}, \mathsf{out}, \mathsf{open}\}$, $Att = \{\{name\}\}$ and $ar(\kappa) = \{name\}$ for $\kappa \in \Sigma$. □

Example 3. Continuing our location model example, we can describe the location state using process expressions. For example, the current state could be:

$$
\begin{array}{l}
\mathsf{building}\{name : \mathtt{itu}\}. \\
\quad \mathsf{floor}\{name : \mathtt{itu3}\}.(\mathsf{room}\{name : \mathtt{3A07}\}.\mathsf{person}\{name : \mathtt{hniss}\}) \\
\mid \quad \mathsf{floor}\{name : \mathtt{itu4}\}. \\
\qquad (\mathsf{room}\{name : \mathtt{4A05}\} \mid (\mathsf{room}\{name : \mathtt{4A09}\}.\mathsf{person}\{name : \mathtt{hilde}\}))
\end{array}
$$

where we have used $\kappa.p$ for $\kappa\{\}.p$ (ie., when $ar(\kappa) = \emptyset$), and where all attributes values are supposed to be constants. □

Next we define process context expressions. Context expressions add *holes* and a *link map* (substitution) to the process expressions.

Definition 4. *For a signature $\Sigma = (\Sigma, N, Att, ar)$ the Σ-*process contexts *W are defined by the grammar*

$$
\begin{array}{lll}
W ::= \sigma \parallel R & & \textit{Σ-process contexts} \\
R \ ::= R \parallel R \mid P \mid 0 & & \text{wide } \textit{process contexts} \\
P \ ::= \kappa\{i : x_i\}_{i\in ar(\kappa)}.P \mid P \,|\, P \mid 1 \mid [\,]_j & & \text{prime } \textit{process contexts}
\end{array}
$$

where $\kappa \in \Sigma$, $x_i \in N$, $j \geq 0$, and $\sigma : N \to N$ is a finite substitution, i.e. the set $dom(\sigma) = \{x \mid \sigma(x) \neq x\}$ is finite.

Define structural congruence for contexts as for processes. □

We introduce a notion of *constants* corresponding to the notion of distinctions found for the π-calculus. The idea is that constant names can not be changed even if the process is placed in a context. We then type process contexts relative to a set of constants $C \subseteq N$ by the rules

$$
\frac{}{C \vdash_\Sigma 0 : I \to 0} \qquad \frac{}{C \vdash_\Sigma 1 : I \to 1}
$$

$$
\frac{}{C \vdash_\Sigma [\,]_j : J \to 1}, j \in J \qquad \frac{C \vdash_\Sigma P : I \to 1}{C \vdash_\Sigma \kappa\{i : x_i\}_{i\in ar(\kappa)}.P : I \to 1}, \kappa \in \Sigma
$$

$$
\frac{C \vdash_\Sigma P : I \to 1 \quad C \vdash_\Sigma P' : J \to 1}{C \vdash_\Sigma P \,|\, P' : I \cup J \to 1} \qquad \frac{C \vdash_\Sigma R : I \to n \quad C \vdash_\Sigma R' : J \to m}{C \vdash_\Sigma R \parallel R' : I \cup J \to n + m}
$$

$$\frac{C \vdash_\Sigma R : I \to n}{C \vdash_\Sigma \sigma \parallel R : I \to n}, \forall x \in C.\sigma(x) = x$$

where I and J are finite sets of natural numbers. We will often omit the C and Σ and simply write $W : I \to J$ when $C \vdash_\Sigma W : I \to J$. A context $W : I \to n$ is *linear* if every index $j \in I$ appears exactly once at a hole $[\,]_j$. We write $W : m \to_L n$ for the linear contexts $W : m \to n$ (where $m = \{0, \ldots, m-1\}$) and also write $W : n$ for $W : 0 \to n$. We will usually omit the map σ if it is the identity on N and in that case say that the context has a trivial link map. In particular, a wide Σ-process expression $r : n$ is regarded as a ground, linear Σ-context expression with trivial link map.

A context $W : I \to n$ can be *inserted* in a context $W' : n \to m$, resulting in the composite context $W' \circ W : I \to m$. In the composition, the names of W are substituted according to the link map of W', and the two link maps are composed. If the context W' is not linear this may imply that some of the sub primes of W are copied and others are discarded. To define composition formally, let $R\sigma$ denote the context $R[\sigma(x_1)/x_1 \ldots \sigma(x_k)/x_k]$, for a wide process context R and substitution σ where $dom(\sigma) = \{x_1, \ldots, x_k\}$. Furthermore, for a wide process context R, let $R[j : P_j]_{j \in I}$ denote the insertion of P_j in all holes of R having index j.

Definition 5. *For contexts $C \vdash_\Sigma W : I \to n$ and $C \vdash_\Sigma W' : n \to m$, define the composite context $C \vdash_\Sigma W' \circ W : I \to m$ by $\sigma' \circ \sigma \parallel R'[j : P_j\sigma']_{j \in n}$ if $W = \sigma \parallel \boldsymbol{\Pi}_{j \in n} P_j$, and $W' = \sigma' \parallel R'$.* □

2.2 Reaction rules

We define reaction rules formally as follows

Definition 6. *For a signature Σ define the set of* parametric Σ-reaction rules *as $PReact_\Sigma = \{(C, R, R', n, m) \mid C \vdash_\Sigma R : n \to_L m, C \vdash_\Sigma R' : n \to m\}$.* □

Given a set of reaction rules $S \subseteq PReact_\Sigma$, the idea is, that a process r can react and become a process r', written $r \to_S r'$ if there exists a rule $(C, R, R', n, m) \in S$, context $C \vdash_\Sigma W$ and a *wide process* parameter $r'' : n$ such that $r \equiv W \circ R \circ r''$ and $r' = W \circ R' \circ r''$. In general, we do not however want all contexts W to allow reactions.

In semantics for process calculi, the contexts that allow reactions are usually referred to as *evaluation contexts*. In the theory of bigraphical reactive systems, evaluation contexts are defined as (linear) contexts over a sub signature $\Xi \subseteq \Sigma$ of *active* prefixes. This captures the evaluation contexts for standard process calculi. For a set S of parametric reaction rules and sub signature $\Xi \subseteq \Sigma$ of active prefixes. we define the set of ground Σ, ϕ-reaction rules by

$$\begin{aligned} React_{S,\Xi} = \big\{\big(L, E \circ R' \circ r\big) \mid\ & L \equiv E \circ R \circ r, \\ & (C, R, R', n, m) \in S, \\ & C \vdash_\Xi E : m \to_L m' \text{ and } r : n\big\} \end{aligned}$$

We say that a process r can react to r', written $r \to_{S,\Xi} r'$ relative to a set of reactions S if $(r, r') \in React_{S,\Xi}$.

Example 4 (CCS and Ambients). The usual CCS reaction rules is then written as the single parametric reaction rule (we use the convention that names $\$n$ denote variables, as opposed to constants):

$$(\emptyset, \mathsf{act}\{ch : \$a\}.[\,]_1 \mid \mathsf{coact}\{ch : \$a\}.[\,]_2, [\,]_1 \mid [\,]_2, 2, 1)$$

It has no constants, it has two holes, and it has width 1. The set of active prefixes is empty, i.e. $\Xi = \emptyset$.

The usual Ambient rule in can be written as the parametric reaction rule $(\emptyset, R, R', 3, 1)$, where

$$R = \mathsf{amb}\{name : \$b\}.(\mathsf{in}\{name : \$a\}.[\,]_1 \mid [\,]_2) \mid \mathsf{amb}\{name : \$a\}.[\,]_3 \text{ and}$$

$$R' = \mathsf{amb}\{name : \$a\}.\big([\,]_3 \mid \mathsf{amb}\{name : \$b\}.([\,]_1 \mid [\,]_2)\big)$$

with active prefixes $\Xi = \{\mathsf{amb}\}$.

We may represent replication by adding a control rep with $ar(\mathsf{rep}) = \emptyset$. □

2.3 Distributed eXtendable processes

We let a *diX-system* be a (partly shared) wide process of width n and a set of *peers* which have each their own signature, reaction rules and evaluation contexts.

Definition 7. *Define a* diX-system *to be a pair* $(r : n, Peers)$*, where* $r = \boldsymbol{\Pi}_{j\in n}\, p_j$ *is a wide process of width* n *and* $Peers = \{peer_i\}_{i\in I}$ *is a set of* peers *of the form* $peer_i = (\Sigma_i, \Xi_i, J_i \subseteq n, S_i)$*, such that* $\forall i \in I.\ \vdash_{\Sigma_i} \boldsymbol{\Pi}_{j\in J_i}\, p_j$*. Reactions of systems is defined by* $(\boldsymbol{\Pi}_{j\in n}\, p_j : n, Peers) \rightarrow (\boldsymbol{\Pi}_{j\in n}\, p'_j : n, Peers)$ *if* $\exists i \in I$ *such that* $\boldsymbol{\Pi}_{j\in J_i}\, p_j \rightarrow_{S_i,\Xi_i} \boldsymbol{\Pi}_{j\in J_i}\, p'_j$ *and* $\forall j \notin J_i . p_j = p'_j$. □

Example 5. A room in the model can be either booked by a person for an activity, or unbooked (independently of whether the room is occupied or not). We model booking status by explicitly maintaining a *status* marker for each room giving us the following process expression:

$$
\begin{array}{l}
\mathsf{building}\{name : \mathtt{itu}\}.\\
\quad \mathsf{floor}\{name : \mathtt{itu3}\}.\\
\qquad \mathsf{room}\{name : \mathtt{3A07}\}.(\mathsf{person}\{name : \mathtt{hniss}\} \mid \mathsf{status}\{bookedby : \mathtt{hniss}\})\\
\mid \quad \mathsf{floor}\{name : \mathtt{itu4}\}.\\
\qquad \mathsf{room}\{name : \mathtt{4A05}\}.(\mathsf{status}\{bookedby : \mathtt{none}\})\\
\quad \mid \quad \mathsf{room}\{name : \mathtt{4A09}\}.(\mathsf{person}\{name : \mathtt{hilde}\} \mid \mathsf{status}\{bookedby : \mathtt{none}\})
\end{array}
$$

The intention now is that a person can book the room he is in *if it is not already booked* by somebody else. This condition describes how coordination is handled in the model:

$$
\begin{array}{l}
\mathsf{room}\{name : \$r\}.(\mathsf{person}\{name : \$p\} \mid \mathsf{status}\{bookedby : \mathtt{none}\} \mid [\,]_1)\\
\longrightarrow \mathsf{room}\{name : \$r\}.(\mathsf{person}\{name : \$p\} \mid \mathsf{status}\{bookedby : \$p\} \mid [\,]_1)
\end{array}
$$

That is, when the condition is satisfied (a person is present in a free room), we simply change the $bookedby$ attribute. Since there may be more than one person in the room we have to ensure that the other persons remain in the room; for this we use holes (matching any number of other persons, even zero, in the room). We use the convention that constant names are written with a true type face, e.g. `none`, and non-constant names are prefixed with a \$ and written in italics, e.g. $\$p$. Thus, for the example the set of constans is $C = \{\mathtt{itu4}, \mathtt{itu3}, \mathtt{3A07}, \mathtt{4A05}, \mathtt{4A09}, \mathtt{none}, \mathtt{hniss}, \mathtt{hilde}\}$.

The distributed nature of the formalism allows each person, presumably each carrying their own device with access to the current location state, to book the room they are in without consulting any other devices. For now, all peers share *one* prime process. Each have a reaction rule as the one above, but personalized to the person associated with the peer, e.g. the peer of `hilde` will have the rule

$$
\begin{array}{l}
\mathsf{room}\{name : \$r\}.(\mathsf{person}\{name : \mathtt{hilde}\} \mid \mathsf{status}\{bookedby : \mathtt{none}\} \mid [\,]_1)\\
\longrightarrow \mathsf{room}\{name : \$r\}.(\mathsf{person}\{name : \mathtt{hilde}\} \mid \mathsf{status}\{bookedby : \mathtt{hilde}\} \mid [\,]_1)
\end{array}
$$

The devices coordinate their actions by ensuring that a room is not booked simultaneously by two persons (the condition above). The obvious problematic situation of two concurrent reactions both seeing a free room, and then updating the location state is handled by the concurrency manager (Sec. 3.4). □

Example 6. Unrelated to the booking of rooms, we may imagine a *position server* keeping track of the locations of client devices. The position server measures the location of clients regularly and adds a client, location pair in an XML document for each measurement. This is done by *out of bands* means, ie., *not* by a reaction rule. Thus, the position pairs can be regarded as *input* to the system. We then make the location state "wide" by having a process p_1 as above, and a process p_2 with location information:

$$\begin{array}{ll} & \mathsf{building}\{name : \mathtt{itu}\}.\ \ldots\ \text{(as before)} \\ \parallel & (\mathsf{pos}\{name : \mathtt{hilde}, where : \mathtt{4A09}\} \mid \mathsf{pos}\{name : \mathtt{hniss}, where : \mathtt{4A09}\}) \end{array}$$

Making the location measures influence the association of people to rooms is a matter of equipping one of the peers in the system (the position server, for instance) with a wide reaction rule for moving persons around:

$$\begin{array}{l} \mathsf{room}\{name : \$from\}.(\mathsf{person}\{name : \$p\} | [\,]_2) \parallel \mathsf{room}\{name : \$to\}.[\,]_3 \\ \parallel \mathsf{pos}\{name : \$p, where : \$to\} \,|[\,]_1 \\ \quad \longrightarrow \ \mathsf{room}\{name : \$from\}.[\,]_2 \parallel \mathsf{room}\{name : \$to\}.(\mathsf{person}\{name : \$p\} \,|[\,]_3) \\ \qquad\quad \parallel 1 \,|[\,]_1 \end{array}$$

Note that the two rooms are separated by a wide parallel composition, allowing rooms to be on different floors.

By distributing the reaction rules to different peers we obtain a minimal (albeit not *enforced*) notion of *abstraction* in the application. We have essentially two systems in play at the same time: a system for booking rooms, and a system for keeping track of the location of people in the building. Those two systems are orthogonal and need not know of each other.

Furthermore, this also provides *openness* because peers may add their own reaction rules and data (as long as they do not change the representation of other peer's data) to the model and the remainder of the system works as expected. ☐

2.4 Relationship to Bigraphs

Linear contexts $W : m \rightarrow_L n$ correspond to (pure) open bigraphs as defined in [11] and their composition is consistent with the definition of composition on bigraphs. However, bigraphs are explicitly typed with *finite* sets of names in the innerface (domain) and in the outerface (codomain). This means that a context $W : m \rightarrow_L n$ would correspond to bigraphs $[\![W]\!] : \langle m, X\rangle \rightarrow \langle n, Y\rangle$ for a choice of finite sets $X, Y \subset N$ such that $dom(\sigma) \subseteq X$ and $\sigma(X) \subseteq Y$, if $W = \sigma \parallel P$. The explicit typing gives control over which names are *not* shared between bigraphs in parallel. This is crucial for the DNF axiomatisation presented in [13] and also for spatial logics for bigraphs presented in [5]. The process calculus presented in the present paper lends itself to the CNF axiomatisation [13], for which one can do without the explicit names and simply assume all names to be shared. A follow up paper will present a fully typed calculus (also including bound names).

3 Implementation

In this section we describe the implementation of the diX-calculus, called Distributed Reactive XML. The implementation is based on *XML Store* [2, 9, 14] and is an extension of the (non-distributed) implementation of *Reactive XML* presented in [16]. XML Store is a general-purpose, peer-to-peer distributed, persistent storage manager for tree-structured data (XML documents). Basing the implementation on XML Store gives a peer-to-peer distributed implementation where it is natural to handle concurrency control by optimistic means.

We start by showing how (prime) process expressions can be represented in XML.

Definition 8. *Assume a signature Σ. Prime Σ-processes are mapped to XML by*

$$\llbracket \kappa\{a_i : x_i\}_{a_i \in ar(\kappa)}.p \rrbracket = \texttt{<}\kappa\ a_1\texttt{="}x_1\texttt{"} \ldots a_j\texttt{="}x_j\texttt{">}\llbracket p \rrbracket\texttt{</}\kappa\texttt{>}$$
$$\llbracket p \mid p' \rrbracket = \llbracket p \rrbracket \llbracket p' \rrbracket$$
$$\llbracket 1 \rrbracket = \epsilon$$

where $\kappa \in \Sigma$, $ar(\kappa) = \{a_1, \ldots, a_j\}$, $x_i \in N$, and ϵ is the empty document. □

Example 7. Rendering the location model process as XML (Ex. 3) gives:

```
<building name="itu">
  <floor name="itu3">
    <room name="3A07">
      <person name="hniss"/>
    </room>
  </floor>
  <floor name="itu4">
    <room name="4A05"/>
    <room name="4A09">
      <person name="hilde"/>
    </room>
  </floor>
</building>
```

□

3.1 System architecture

XML Store is a *storage manager* for tree structured values (data)—concretely, XML documents. Stored values can later be retrieved via XML Store. The interface only allows one to specify *what* to store, not *where*. Therefore the XML Store implementation is free to move stored values about. Once stored, a value is identified by a location-independent identifier (typically, a cryptographic hash of the contents of the value).

Though the XML values, representing processes, themselves do not have to be distributed, it makes sense to do so. XML Store provides wide-scale distribution of the values it is storing by using a peer-to-peer routing algorithm (the current implementation uses Kademlia [12]). This distribution is built into XML Store, hence relieving the application programmer of implementing his own distribution layer. Distribution in XML Store is transparent so an application cannot observe whether a value is stored locally or remotely.

The basic architecture of Distributed Reactive XML is an XML Store distributed over a number of peers, which provides clients with access to the current process. To the application programmer this appears to be just an XML Store. Clients connect to this XML Store either by joining the peer-to-peer network, or as traditional clients. Since one could imagine different situation where each of them would be an advantage, it makes sense to have both options. For instance, the Position Server which updates the current process on a regular basis would most likely benefit from being a part of the network, instead of connecting to the XML Store each time an update takes place. On the other hand, clients with less resources, for instance PDAs, may not have resources available to join a peer-to-peer network, and they would therefore connect to the XML Store as clients.

Figure 1 shows a setup with four clients. Each client has its own set of reaction rules (Sec. 2.2) and a handle to the shared process expression.

3.2 Implementing reactions

For simplicity, we will only consider prime reaction rules, that is, reaction rules $(C, R, R', n, 1)$. This means that we only need to consider evaluation contexts with one hole and that reaction are always performed inside the same prime process. Performing a reaction $p \rightarrow_{\Sigma,\Xi} p'$ then amounts to finding a reaction rule $(C, R, R', n, 1) \in S$, an evaluation context $C \vdash_{\Xi} \sigma \parallel R_E : 1 \rightarrow 1$ and a wide process expression $r = \boldsymbol{\Pi}_{j\in n}\, p_j$ such that

$$p = R_E[R[j : p_j]_{j\in n}\sigma] \tag{1}$$

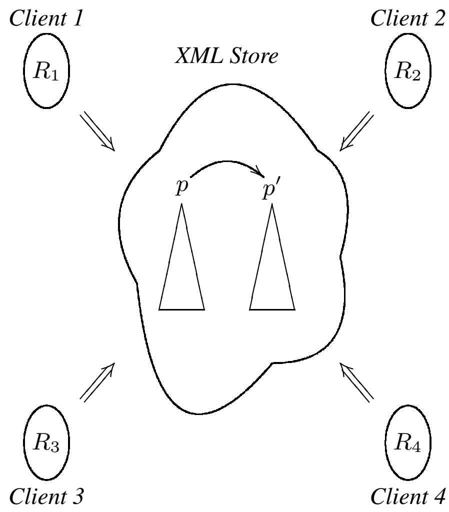

Fig. 1. Distributed Reactive XML setup.

and then compute $p' = R_E[R'[j : p_j]_{j \in n}\sigma]$.

We use XPath expressions to determine evaluation contexts.

Definition 9. *For a prime Σ-process p and an XPath expression ϕ, let xpath(ϕ, $[\![p]\!]$) denote the set of roots of subtrees in $[\![p]\!]$ that satisfies ϕ.* □

For a sub signature $\Xi \subseteq \Sigma$ define the XPath expression

$\phi_{\Xi,\Sigma} =$ `//*not(ancestor-or-self::*`
`[name()='`κ_1`' or name()='`κ_2`' or ... name()='`κ_k`'])]`

for $\Sigma \backslash \Xi = \{\kappa_1, \ldots, \kappa_k\}$. Then $xpath(\phi_{\Xi,\Sigma}, p)$ determines the roots of subtrees p' of p such that $p = E \circ p'$ and E is an evaluation context.

Now note that $R_E[R[j : p_j]_{j \in n}\sigma] = R_E[R\sigma[j : p_j\sigma]_{j \in n}] = R_E \circ R\sigma \circ r\sigma$. For any process r' and σ there exists a process r such that $r' = r\sigma$. Thus, solving equation (1) amounts to finding a complete subtree $t_R = R\sigma \circ r'$ in p for some r' and substitution σ such that the subtree t_R has a root that is a child of a node in $xpath(\phi_{\Xi,\Sigma}, p)$.

To find a sub tree $t_R = R\sigma \circ r'$ in p for some wide process r' and substitution σ we search for the context R up to a possible substitution σ (computed as constraints during the attempted match) of the names in R, and allowing the holes in R to match any prime process, even an empty tree (i.e. a nil process 1). If the context $R\sigma$ is found for some substitution σ, and prime processes p_j matched with holes, it is checked if the root of the context $R\sigma$ belongs to the solution set of the XPath expression $\phi_{\Xi,\Sigma}$. If so, the matching algorithm reports back the substitution σ, the root of the context $R\sigma$ and the (roots of the) sub prime processes p_j matched with holes. This is a generalisation of the standard (ordered) sub tree problem for trees. As for the standard problem the matching algorithm is extended to unordered trees by using a bipartite matching algorithm each time a set of children in the pattern R is matched against a set of children in the source tree p. To perform the reaction all that is needed is to replace sub tree t_R in p with $R'\sigma[j : p_j]_{j \in n}$.

3.3 Distributed reactions in XML Store

The processes stored in XML Store are values. This means that a process, once stored, does not change; in other words, it is *immutable*. Since a value is never updated we can freely cache it at (copy it to) all interested parties. It also means that we have to take special measures to perform the equivalent of updates on the process. Instead of destructively updating the value, we compute a *new* value with references to unchanged parts of the old value. In other words, we share (parts of) the stored values. This sharing is transparent to the application programmer [2]. A consequence of this is that XML Store really stores DAGs rather than trees. The "newest" value (the current state of the system) is bound to a *handle* (in practice through a name service) which *can* be updated.

From this "value-oriented" perspective, the steps a client performs to realize a reaction are as follows:

1. *Find all posssible redexes by finding all evaluation contexts.*
   For our example location system, we allow reactions on all (sub-) processes, and therefore the XPath expression locating evaluation contexts will simply select all nodes (`//*`). Performing this on the process in Ex. 7 will return a set containing all nodes.
2. *Match each of the possible redexes against the left hand side of the reaction rule instantiating holes and variables.*
   Matching all nodes in the process against the left hand side of the "book room"-reaction rule (Ex. 5), will result in a match between the left hand side of the reaction rule and room `4A09` with variables $\$r$ instantiated to `4A09` and $\$p$ to `hilde`, and the hole $[\,]_1$ mapped to 1.
3. *If any match exists, the reaction can be executed by calculating a reactum based on the right hand side of the reaction rule, and reconstructing the process expression.*
   Since all data stored in XML Store is immutable, clients cannot simply change the matched node (the redex) in the process tree to reflect the changes. Instead they have to build up a new tree. Fig. 2 illustrates this situation. Before the reaction, the process is as seen in Fig. 2(a). After the reaction, Fig. 2(b), a new process has been built, but new nodes have only been constructed from the nodes which have to be "updated" (the reactum) up to the root. On the path unchanged nodes are reused.

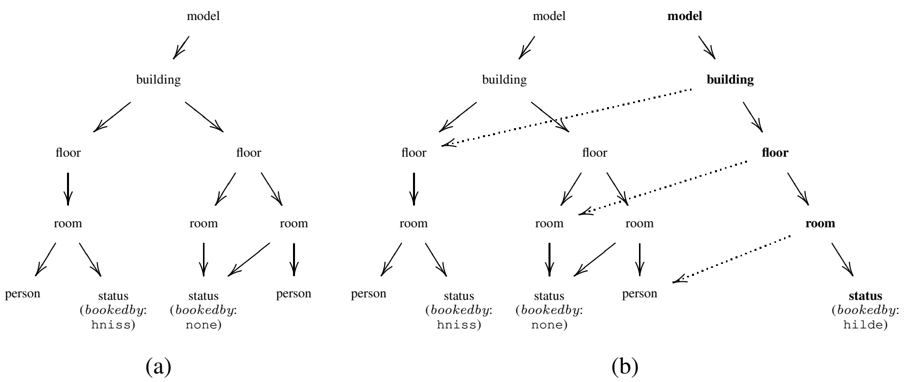

Fig. 2. Reusing unchanged nodes (dotted arrows indicate reuse; bold text indicate the newly constructed path).

The handle to the current process will at this point still refer to the old root node p. To make other clients aware of the new process, the client has to updated the handle to the new root p'.

Such updates of handles (the only updates possible with XML Store) are done using an atomic compare-and-swap algorithm, which guarantees that nobody has changed the value in the time $\Delta t = [t_{read}; t_{swap}]$. By using this facility, we are able to obtain a simple distribution of client updates to the process. Thus ultimately, this is how coordination is implemented.

3.4 Synchronizing updates

The simple form of synchronization mentioned above works, but does not support situations where several clients simultaneously inspect the current process, find possible reactions, and build up a new process. To handle this, we will allow non-conflicting reactions (intuitively, reactions in *different* parts of the process) to take place concurrently. We use the term *conflicting reactions* to denote the situation where we are not able to incorporate changes from two (or more) reactions without leaving the process in an inconsistent state.

Assume that the two reaction rules $\mathcal{R}_1$ and $\mathcal{R}_2$ are performed on the same process. The reactions are performed simultaneously, consequently, they will inspect the process in the exact same state. We can now state two situations with conflicting reactions:

1. The two reactions overwrite each other's changes. Since they are both changing the same nodes, we cannot fuse the changes from both reactions to one process tree.
2. One (or both!) of the reactions makes changes to the redex for the other reaction. Since a reaction is only possible if the rule matches the redex, this situation removes the initial condition for one or both of the reactions.

As described in Sec. 3.2 performing a reaction on the process p, amounts to finding a matching subtree (a redex) t_R in p and replace this with $R'\sigma[j : p_j]_{j\in n}$. Assume now that when performing $\mathcal{R}_1$, a subtree t_{R_1} in p is found. Additionally, a subtree t_{R_2} is found for $\mathcal{R}_2$ in p. We know that all nodes changed when performing $\mathcal{R}_1$ must be within the subtree t_{R_1}, and all nodes changed when performing $\mathcal{R}_2$ must be within the subtree t_{R_2}. Hence, a conservative estimate for non-conflicting reactions are: if $\mathcal{R}_1$ does not change any nodes in t_{R_2} and likewise $\mathcal{R}_2$ does not change any nodes in t_{R_1}, the two reactions will not have any overlapping changes.

Definition 10. *If $subtree(n)$ is a function $subtree : Node \rightarrow Set(Node)$ returning a set containing all nodes in the tree with root n, t_{R_1} is the redex for the reaction $\mathcal{R}_1$ performed on p and t_{R_2} is the redex for the reaction $\mathcal{R}_2$ performed on p. The two reactions $\mathcal{R}_1$ and $\mathcal{R}_2$ are conflicting, if $subtree(t_{R_1}) \cap subtree(t_{R_2}) \neq \emptyset$*

We can use this knowledge in an optimistic concurrency control manager, where we allow clients to inspect the process expression at any time. The client will then find possible reactions. When it is ready to commit the result of one of these reactions, we *validate* whether the reaction is in conflict with other reactions performed in the time between the client inspected the process and the attempted commit operation. If any reactions occured, for each of them we check that the redex for that reaction does not have any nodes in common with the redex for the reaction we are about to commit. If there are no conflicts, we can incorporate the changes from this reaction in the shared process. In case of conflicts, we simply abort the commit operation.

In order to be able to do this validation, we need to track each reaction performed and the matching subtree (redex) that was the condition for the reaction. We capture these in so-called versions. A *version* consists of the *resulting* process tree and a changeset. A *changeset* records the changes that takes the original process tree (before the reaction took place) to the process tree stored in the version. Therefore, a changeset consists of the redex, the resulting reactum, and a XPath expression indicating what part of the process tree was rewritten.

Example 8. Consider again the reaction where `hilde` successfully books room `4A09`. In that case the version contains the process tree depicted in Fig. 2(b) and a changeset. The changeset contains the redex

$$\mathsf{room}\{name : \texttt{4A09}\}.(\mathsf{person}\{name : \texttt{hilde}\}|\mathsf{status}\{bookedby : \texttt{none}\}),$$

the reactum

$$\mathsf{room}\{name : \texttt{4A09}\}.(\mathsf{person}\{name : \texttt{hilde}\}|\mathsf{status}\{bookedby : \texttt{hilde}\}),$$

and an XPath expression indicating which room was booked:

```
/child::*[1]/child::*[2]/child::*[2]
```
□

We can now describe what is really stored in the XML Store, namely the latest version together with a list of versions leading to that version. The aggresive use of sharing in XML Store avoids the obvious problem of repeatedly storing the same (parts of) process trees again and again.

As a side effect of storing changesets, we are able to track all changes on a reaction-by-reaction basis. This gives us a nice feature for debugging ReactiveXML.

3.5 Implementation details

Distributed Reactive XML, as described above, has been implemented (in Java) using the features provided by XML Store. The implementation covers the complete Distributed, eXtendable Process Calculus; for example, the process expressions and reaction rules for our running example have all been executed with the implementation. For this to work in practice, we have integrated the system with a position server, Ekahau [7], that positions Wireless LAN clients. The integration lets the position server update directly the queue of positions events, but in a safe manner so that the updates include appropriate changeset information. To the other peers, therefore, this looks like the result of executing any other reaction rule.

The implementation and the location model example is available on the web: ⟨`http://www.itu.dk/research/theory/bpl/reactivexml/`⟩.

4 Conclusion

We have shown how one can utilize the similarities of XML and the theory of bigraphs to implement an open, distributed XML-based coordination middleware, having a simple distributed eXtendable process calculus as formal foundation that encompasses both the shared data, processes and reaction rules. The implementation was based on a so called *value-oriented, peer-to-peer XML Store* previously implemented at ITU and DIKU. We demonstrated how the value-oriented approach facilitates a cheap implementation of optimistic concurrency control in which complete histories of processes are stored. Finally, we have exemplified the use of the coordination middleware by a location-based service system, which has been implemented and is running at ITU.

Future work: Many tasks remains for future work. We are currently working on extending the diX-calculus to cover binding bigraphs. This includes allowing local and bound names, as well as introducing explicit finite names sets on the interfaces. Name binding is reflected in the XML-representation as IDREF and ID values of attributes. We also work on implementing wide reaction rules. We intend to investigate if one could use XML query-languages, such as XQuery or TQL in the implementation of matching. We also consider how to extend the diX-calculus such that peers can enter and leave dynamically and can change their access to the shared state, as well as the benefits from making the reaction rules part of the shared process data. We also plan to investigate the applications of the general bigraph theory. One application could be to use the theory of bisimulation to prove that the concurrency control is correctly implemented. Another direction is to investigate the uses of the spatial logics for bigraphs as developed in [6]. Finally, we consider how to treat security in this setting, e.g. by allowing cryptographic functions to be computed on the values during reactions or exploiting the complete histories stored in XML store.

Related work: The recent paper [5] reports on independent work relating bigraphs and XML. However, the focus of [5] is on representing XML-data as bigraphs and the use of bigraph-logics [6] to describe properties of XML-data. This is opposed to the present work, in which we exploit XML technologies (XPath and XML Store) for the implementation of (bigraphical) reactive systems as XML. The paper [8] introduces the process calculus $Xd\pi$ based on the π-calculus aimed for modelling XML-centric peer-to-peer systems and investigates its bisimulation semantics. It would be interesting to try to represent the $Xd\pi$-calculus in diX and e.g. compare the general bigraph bisimulation semantics to the one for $Xd\pi$. Active XML [1] provides a language and foundation for *active* XML documents. Active XML documents support dynamic inclusion of XML data produced by web-services, which possibly could be used jointly with Distributed Reactive XML.

Bibliography

[1] Serge Abiteboul, Omar Benjelloun, Ioana Manolescu, Tova Milo, and Roger Weber. Active XML: Peer-to-peer data and web serivces integration (demo). In *Proceedings of the international VLDB Conference*, 2002.

[2] Thomas Ambus. Multiset discrimination for internal and external data management. Master's thesis, Dept. of Computer Science, University of Copenhagen (DIKU), 2004. URL ⟨`http://www.thomas.ambus.dk/plan-x/msd/`⟩.

[3] Luca Cardelli. Semistructured computation. In Richard C. H. Connor and Alberto O. Mendelzon, editors, *Research Issues in Structured and Semistructured Database Programming. Proceedings of the 7th International Workshop on Database Programming Languages (DBPL'99), Invited Paper*, volume 1949 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag GmbH, 2000.

[4] P. Ciancarini, R. Tolksdorf, and F. Zambonelli. Coordination Middleware for XML-centric Applications. In *Proc. ACM/SIGAPP Symp. on Applied Computing (SAC)*. ACM Press, 2002. URL ⟨`citeseer.ist.psu.edu/ciancarini02coordination.html`⟩.

[5] Giovanni Conforti, Damiano Macedonio, and Vladimiro Sassone. Bigraphical logics for XML. 2004.

[6] Giovanni Conforti, Damiano Macedonio, and Vladimiro Sassone. Bilogics: Spatial-nominal logics for bigraphs. 2004.

[7] Ekahau. Ekahau Positioning Engine 3.0: Developer's Guide. URL ⟨`http://www.ekahau.com/`⟩. 2004.

[8] Philippa Gardner and Sergio Maffeis. Modelling dynamic web data. In Georg Lausen and Dan Suciu, editors, *9th International Workshop on Database Programming Languages (DBPL'03)*, volume 2921 of *Lecture Notes in Computer Science*, pages 130–146. Springer-Verlag GmbH, 2003.

[9] Fritz Henglein and Henning Niss. Plan-X webpage, 2005. URL ⟨`http://www.plan-x.org/`⟩. (XML Store is part of the Plan-X Project).

[10] Kuen-Fang Jack Jea, Shih-Ying Chen, and Sheng-Hsien Wang. Concurrency control in xml document databases: Xpath locking protocol. In *ICPADS*, pages 551–556. IEEE Computer Society, 2002. ISBN 0-7695-1760-9.

[11] Ole Høgh Jensen and Robin Milner. Bigraphs and Mobile Processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge, Computer Laboratory, February 2004. URL ⟨`http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-580.pdf`⟩.

[12] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 53–65, 2002. URL ⟨`http://www.scs.cs.nyu.edu/~dm/papers/maymounkov:kademlia.ps.gz`⟩.

[13] Robin Milner. Axioms for bigraphical structure. Technical Report UCAM-CL-TR-581, University of Cambridge, Computer Laboratory, 2004. URL ⟨`http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-581.pdf`⟩.

[14] Kasper Bøgebjerg Pedersen and Jesper Tejlgaard Pedersen. Value-oriented XML Store. Master's thesis, IT University of Copenhagen, 2002. URL ⟨`http://www.it-c.dk/people/kasperp/xmlstore/pdf/thesis.pdf`⟩.

[15] Jean-Bernard Stefani. Requirements for a global computing programming model. URL ⟨`http://mikado.di.fc.ul.pt/repository/D1.1.2v1.0.pdf`⟩. Mikado Deliverable D1.1.2, 2003.

[16] Jacob W. Winther. Reactive xml. Master's thesis, IT University of Copenhagen, 2004.