Exploiting XML for Coordinable Document Agents

Paolo Ciancarini
DSI - University of Bologna
ciancarini@cs.unibo.it
http://www.cs.unibo.it/~cianca

Franco Zambonelli
DSI - University of Modena and Reggio Emilia
franco.zambonelli@unimo.it
http://www.dsi.unimo.it/Zambonelli

Outline of the Tutorial

- Introduction
  - the Internet scenario
  - motivations and goals of the tutorial
- Part 1: coordination of active document agents
  - active documents
  - mobile agents
  - coordination models and languages
- Part 2: XML-based approach for coordinable document agents
  - XML agents and active documents
  - XML middleware
  - XML coordination laws
The Internet Scenario

- "The Computer is the Network"
  - the convergence of Information and Communication technologies process new opportunities for industry, research, and teaching
- The Internet as a global distributed computing system
- Pushing the development of novel appliances, applications, services, and organizational theories
  - traditional distributed applications executed at a world-wide scale: CSCW, DBMS, grid computing.
  - brand new application areas: complex web services, information retrieval, E-commerce, etc.
  - emerging scenarios: mobile and embedded computing, wearable computing systems, networked homes
  - emerging organizational and economic models (e.g. the Napster’s impact on music industry)

The Internet Challenge

- Challenging traditional computing models and engineering principles
  - decentralization, unreliability, dynamicity
  - heterogeneity
  - network-awareness and mobility
  - from “directed” to “supervised” computing
  - from “programmed” to “emerging behaviors”
- Need for computer scientist and engineering to:
  - adapt themselves to the new platform
  - invent new methods to exploit the new computing models enabled by the Internet
- Shift from “Computer Science” to “Internet Science”
Motivations of the Tutorial

- The Internet is “Document Centric”
  - several Internet applications deals with document exchanges (e.g., the basic HTTP as well as napster)
  - CSCW typically deal with accesses to shared workplaces or document spaces
- Document-centric computing models and needed
  - shift from process- and object-oriented computing to document-oriented computing
  - documents enriched with computational activities and intrinsically mobile
  - relations with agent-based computing
- Focus on two key issues:
  - coordination models and infrastructures for the engineering of complex active documents applications
  - XML as “The Standard” for active documents

Goals of the Tutorial

- Give a general and unifying characterization of
  - documents
  - active documents
  - software agents
  - mobile agents
- Introduce the main concepts underlying
  - coordination models
  - coordination languages and infrastructure
  - the use of the above in active document applications
- Outline the role of XML in above contexts
- Survey some key proposals in the area
Part 1: Active Documents and Coordination

- What are active documents?
  - Passive documents
  - Re-active documents
  - Pro-active documents and Software agents
  - Applications of re-active and pro-active documents

- Mobile Agents vs. Active Documents
  - document mobility
  - agent mobility

- Coordination of Document Agents

- Coordination models and languages

---

Passive Documents

- From a computer science perspective
  - a document is any kind of data structure that applications can exchange with each other and elaborate

- Documents as data structures:
  - by definition – documents must have a content, i.e., must contain “information” or DATA
    - text, images, music, anything....
  - In addition, they can have a “representation” or a “structure”
    - tags, format information, etc....

- Isomorphism between documents and data structures
Representing Documents

- **(passive) document = content + representation**
- Documents contain meta-level (or **structural**) information
  - needed for external entities (e.g., humans or programs) to **understand** the information
    - Examples: Glossary in books, Header of a BMP file
  - needed for external entities (e.g., humans, search engines, or printers) to properly **elaborate** the information
    - Examples: tags in HTML files, Tex documents
- **Two main approaches**
  - binary, proprietary, representation (e.g., MSWord, PDF)
  - open, ASCII based, representations

Marking Up

- Open document representation are ASCII-based
- ASCII documents are “flat”, and their representation is either:
  - wired into applications that parse and manage them
  - given by “marking” the document with some “code”
- Examples: Tex, HTML …
- Such “Formatter Programs” assign a rendering behavior to documents
  - mark-ups are interpreted as commands
  - to build a “page”, they are driven by markup commands interspersed in the document text
  - they are like compilers...
**Procedural vs. Declarative Markup**

- Several markup languages add information to specific fragments of a document
  - HTML: `<i>hi, my name <b>Franco</b> is in Italic</i>`
  - TeX: `hi, my name \textit Franco \normal is in Italic`

- They are only procedural:
  - the logical structure of the document is not expressed
    - searching document abstractions (e.g., titles and sections) is difficult
    - the concept of “style” is non-existent

- Declarative markup languages add (i.e., declare) “structural” information:
  - *Latex*: `\documentstyle[twocolumn]{article}`

---

**Declarative Markup: SGML**

- Standard generalized declarative markup language, proposed by IBM in 1980 and adopted in 1986 by ISO

- A SGML document is composed of three parts
  - SGML declaration
  - Document Type Declaration
  - Document instance.

- SGML specifies
  - the structural elements composing a document
  - not the rendering of its documents

- SGML includes a meta-language to declare new tag types, to form a Document Type Declaration (DTD)
Declarative Markup: XML

- Standard declarative markup language, defined by W3C, to describe the structure of documents
  - as a simplification of SGML toward Web-based document processing
- REMARK: Declarative markup is either structural or semantic
  - Structure: A book is composed of chapters, sections, titles, notes, etc. A letter is composed of sender, addressee, salutations, body, signature, attachments, etc.
  - Semantics: A news item about a criminal act may specify the source of the news item itself, the description of a sequence of acts, the name of the place where the acts took place, the name and rank of the involved police officers, the stolen amount, etc.

XML Example

```xml
<?XML version="1.0"?>
<!DOCTYPE CourseEntry SYSTEM "http://uni.edu/UniCrs.dtd ">
<course>
  <coursename>Computer Networks</coursename>
  <year>4</year>
  <semester>1</semester>
    <teacher>Jane Smith</teacher>
  <lesson>
    <lessonname>Introduction</lessonname>
    <lessonnumber>1</lessonnumber>
    <abstract>blah blah</abstract>
    <reading>
      <authors>..</authors>
      <title>..</title>
      <book>..</book>
      ...
    </reading>
  </lesson>
</course>
```
Towards Active Documents: HTML vs. XML

- The rendering of HTML documents is up to the browser
  - there is no specific “behavior” associated to a document
  - different browser can handle differently different tags
- The rendering/manipulation of XML documents can be associated to the XML document
  - XML tags are not associated to any predefined behavior
  - XSL stylesheet associated to a document (or to a document type) can be used to associated a behavior to XML documents
  - XSL rules the rendering/manipulation of a document
- $XML + XSL = content + structure + behavior$

What is an Active Document

- Active document = content + (structure + behavior)
- The document “knows” how to be handled
  - its structure can be subject to document-specific behaviors
- The possible perspectives:
  - external applications are forced to use the internal agent behavior to handle documents
    - REACTIVE DOCUMENTS
  - the agent can handle itself in autonomy
    - PROACTIVE DOCUMENTS
- Several approaches possible….
Scripting Documents

- Scripting documents to obtain specific – reactive – behaviors of the documents
  - scripting is executed upon user events
  - full power of scripting actions

- The structure is not necessary (procedural markup)
  - JavaScript: does not necessarily apply to the document structure, can be associated to any portion of a document

```
<a HREF="provejs.html" onClick="var n = 1;
window.open('http://www.dsi.unimo.it');
n = n + 1;
alert('Apro finestra numero ' + n);
window.open('http://www.dsi.unimo.it')">
  DSI
</a>
```

Object Documents

- A (part of a) document is an object
  - content (+ object attributes) represents the object data structure
  - methods of the documents specify how to handle the data structure
  - scripting may be used to program how to invoke object methods in reaction to user events

- Interesting properties
  - interface of documents
  - state of documents (memory of past activities)

- Reactivity not proactivity
Compound Documents

- A document is a nested structure of objects
  - each object handles portions of a document
  - the different parts of the structure of a document are to some extent independent, but they can somehow interact with each other
  - e.g., container objects control inner objects
- “heterogeneous documents”
  - different components in a documents (e.g., text, figures, charts, etc…)
  - needs to be handled independently and in different ways
- Promotes mobility and restructuring
  - each part of the document know how to be handled
  - can be moved across documents

Example: ActiveX Control & VBScripts

- A text label that changes on clicks

```xml
<OBJECT ID="Label1" WIDTH=104 HEIGHT=27
  CLASSID="CLSID:99B40-6EC7-11CF-A6C7-00AA">
  <PARAM NAME="_ExtentX" VALUE="2752">
  <PARAM NAME="_ExtentY" VALUE="714">
  <PARAM NAME="Caption" VALUE="Label Cntrl">
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
  Sub Label1_Click()
    Dim a
    a = Label1.Caption
    If a = "Start" Then
      a = "Stop"
    Else
      a = "Start"
    End If
    Label1.Caption = a
  End Sub
</SCRIPT>
```
Objects vs. Documents

- Objects = data + (implicit) structure + behavior
  - the accent is on providing “services”
  - clients (e.g., other objects or users) request services
  - objects execute services and answer at service completion

- Reactive documents = data + (possibly explicit) structure + behavior
  - the accent is on providing “content”
  - clients (e.g., other objects or rendering applications) request “content” (e.g. by clicking on a link or on a part of a document)
  - documents (possibly via the mediation of scripting) execute specific internal services to provide the needed content (in the needed form) thereafter

Pro-Active Documents

- The documents handles itself in autonomy
  - autonomous execution threads inside the documents
  - does not require external events to execute (only the event that “put the document to life”

- The trivial example: Applet Documents
  - the content of a document can be embedded into an applet
  - the applet start as soon as the applet document is loaded into a browser
  - the applet (and only the applet) can autonomously handle the document content
  - still capable of reacting to event (proactivity does not exclude reactivity)
Active Documents and Software Agents

- Agents as autonomous software entities
  - have their own thread of control/execution
  - proactive behavior
  - may be “intelligent”

- Weak notion of agency
  - does not focus on “intelligence”, but rather on the fact that an agent, unlike an object, is not simply a “service provider”, but rather a “task-oriented entity”, which execute to achieve a specific goal

- Agents = data + behavior + autonomy
  - accent on autonomy of computation toward a goal

- Active documents = content + behavior + activity
  - accent on data and on autonomous activity on data manipulation

Object vs. Agents

- Objects are somehow “passive” or simply “re-active”
  - their execution is activated upon request, and within the same execution flow of the invoking client

- Object-based vs. Agent-based Applications
  - Object-based
    - global goal achieved via a global control scheme for the application entities
    - design by delegation of control
  - Agent-based applications
    - sub-goals assigned to autonomous agents (integrating execution capabilities, i.e., threads) which try to achieve in autonomy their own goal
    - design by delegation of responsibility
Passive vs. Proactive Documents

- If documents are somehow “passive” or simply “reactive”
  - they are activated under the control (and often within the same execution thread) of an external entity

- Passive vs. Proactive Document Applications
  - Passive and Reactive Documents
    - external entities control when to invoke documents behaviors
    - design by delegation of activity specification
  - Proactive Documents
    - documents control how and when to behave in a certain way
    - design by delegation of activity

- Let’s talk of DOCUMENT AGENTS
  - to better characterize proactive agents

Applications of Document Agents (examples)

- The Active Agenda
  - storing dates and times of meetings
  - re-scheduling meetings on the basis of newly discovered constraints
  - when a new meeting among a group is to be organized: the agenda activates, and by cooperating with the other agendas of the persons in the group, tries to find a suitable schedule

- The Active Portfolio
  - storing data about one person’s (or one group’s) investments
  - monitor selected stock exchanges
  - signal relevant events to users
  - take initiative on buying and selling of shares
Mobility of Documents

- **Document Mobility**
  - Mobility is intrinsic in information
    - we produce information only via mobility

- **Mobility is intrinsic in the Web**
  - standard HTML data representation to enable mobility
  - standard HTTP protocol to actually move documents
  - information + meta-information mobility

- **Mobility of document agents adds a further component**
  - mobility of behaviors together with information
    - Examples: HTML Scripting and Applets
  - mobility of execution state
    - Example: the thread of an applet that stops, save itself, and restart on a different browser

Agents Mobility

- **Mobile Agents**
  - can roam across Internet sites
  - carrying along:
    - portable code, portable execution state, portable data

- **Types of mobility**
  - re-active (you must go to)
  - pro-active (I must go to)
  - physical (I go where my device go)

- **Reasons for agent mobility**
  - save bandwidth (move computation local to data)
  - reliability (no continuous network connection needed)
  - suitable engineering paradigm for wide-area computing
  - software components on mobile devices are mobile agents
Documents vs. Agent Mobility

- **Document Mobility**
  - accent on data mobility
    - the information in the documents gets transferred
    - but the behavior of the document and its execution state is moved too!!!

- **Agent Mobility**
  - accent on computation mobility
    - the code and the execution state of the agent moves
    - but data of the agent is moved too!!!

- **Two facets of the same paradigm**
  - need portable code
  - need safe code execution
  - need safe and secure movements of data

XML and Mobility

- **XML data representation is intrinsically mobile**
  - machine and human readable code
  - not tied to any specific application

- **XSL specifications**
  - representing rules for document transformations
  - again portable

- **Java behaviors**
  - XSL can specify Java classes to code specific behavior and transformation rules for a document
  - Java classes are intrinsically mobile (and dynamically downloadable)

- **Mobile Document Agents can take advantage of XML!**
Application Example: Searching Related Documents on the Web

- Application implemented with the Adlet active document systems
  - given one document, how can we effectively search – in background – for related documents
- Any Adlet document contains, in addition to the content:
  - a descriptive part (e.g., keyword, structure, etc.)
  - a behavioral part
    - documents proactively move across the Web (according to specific policies)
    - document interact and communicate with other documents to discover relationships
- Once the goal (finding enough documents) is achieve, the Adlet go back to its home

Application Example: Reconstructing a Map

- The scenario of a disaster
  - a team of people exploring the environment and
- Document agents on mobile devices carried by each team member
  - store info about the environment
  - dynamically monitor the environment to acquire new data
  - reconstruct a local map of the environment
  - coordinate themselves with the document agents to reconstruct a global map from the local one
- Mobility of document agents
  - fixed on the devices
  - mobile w.r.t. to the context they perceive and in which they execute
Interactivity of Document Agents

- Document agents, despite autonomy
- may be in need to acquire info from external sources
  - needed to complete their task
    - Example: an applet receiving data from the server
    - Example: an active document retrieving info from a DBMS
- may be in need to synchronize their action
  - with external users/applications
    - Example: waiting for events
- may be part of a multi-document (compound document) system
  - exchange data and coordinate actions
    - Example: a document coordinating with related documents
- Handling all this interaction and coordination issues is part of the overall activities of document agents
  - software engineering issues
  - mobility issues (mobility challenges interactions)

Software Engineering Document Agents Applications

- When distributed, multi-component applications are to be designed and developed
  - need to define inter-component relationship
  - need to define “the software architecture”
  - engineered approach to multi-component application development
- Suitable abstractions and frameworks are needed
  - to build coherent and well structured systems from single components
  - to understand, manage, maintain, complex multi-component applications
Traditional vs. Agent-oriented SW Engineering

- Traditional, object-based and component-based, software engineering:
  - accent on “composition” rather than on “interaction” and “coordination”
  - static software architectures
  - no suitable conceptual tools to deal with dynamicity of interactions (mobility, autonomous behavior, openness)

- For active entities, like agents and document agents
  - agents do not compose with each other but “interact” and “coordinate” with each other
  - their behavior, as well as their operating environment, is dynamic

- Need of suitable tools for handling "Coordination" in an open, dynamic, application scenario

What is Coordination?

- "Coordination is managing dependencies between activities" (Malone & Crowston)
  - interdisciplinary issue

- "Coordination is constraining the interaction" (Wegner)
  - CS viewpoint

- Coordination is the process of building programs by gluing together active pieces" => “A coordination model is the glue that binds together activities into an ensemble” (Carriero & Gelernter)

- “A coordination model provides a framework in which the interaction of active and independent entities … can be expressed" (Ciancarini)
  - how to express interaction
The Interaction Space

interaction space:

Communication, synchronization, collaboration, competition etc..

Coordination Models, Languages, and Infrastructures

- coordination model: the formal basis (semantics) for a coordination language
- coordination language: a set of language constructs to be combined with a conventional programming language to obtain a fully-fledged programming language
- coordination infrastructure: the “middleware” layer that, in distributed systems, support the coordination activities of distributed (possibly mobile) components

Several models, languages, and infrastructures define in the previous years
  - several open issues for agent-based and document-based coordination!
What is a Coordination Model?

- A conceptual (formal) framework to model the space of interaction
  - useful to study and understand problems in designing programming languages and software architectures including several agents
- defines how agents interact and how their interactions can be controlled:
  - This includes dynamic creation and destruction of agents, control of communication flows among agents, control of spatial distribution and mobility of agents, as well as synchronization and distribution of actions over time
- Coordination models differ mostly in the way they control interaction

An Ontology for Coordination Models

- A coordination model is triple $<E, M, L>$ where
  - $E$ are the coordinables
    - the entities (agents, processes, objects, components, and so on) whose mutual interaction is ruled by the model
  - $M$ are the coordination media
    - the abstractions that aggregate (virtually or actually) coordinables into an ensemble, and that enable and rule their interactions
  - $L$ are the coordination laws
    - the rules governing the interaction among coordinables as well as the behavior of the coordination media
    - define the semantics of a number of coordination mechanisms that can be added to a language
Coordinables

- A coordination model defines
  - what is a coordinable to it
- No interest in the coordinable’s inner structure
  - coordination concerns interaction
- Definition on the boundary
  - observable behavior
- Admissible coordination entity
  - any entity expressing through the model’s coordination languages
  - they must respect the syntax of the coordination language to interact

Coordination Media

- Populating the interaction space
  - communication channels, monitors, connectors, blackboards, …
- A coordination model defines
  - what is a coordination medium to it and how agents are aggregated in a coordination medium
- Full definition
  - observable & inner behavior
  - state transitions
- Given a model, the coordination media can be organized according to different architectures
  - a multiplicity in number/kind?
  - global or local, private or public?
  - physically distributed/centralized?
  - …
Coordination Laws

- Rules governing the space of agent interaction
- Defined in terms of
  - semantics of the coordination languages
- Determined by
  - coordination media behavior
- Given a model, are the coordination laws
  - completely fixed a-priori by the model?
  - generally fixed but to be specialized on need?
  - fully programmable?
  - implicitly/explicitly represented?
  - dynamically modifiable?
  - physically distributed/centralized?
  - ...

Coordination Models: Ruling the Interaction Space

interaction space = coordination media + coordination laws
Example: Tuple-based coordination models (i)

- Based on the concept of associative access and synchronization via shared dataspaces
- Derived from Linda (Gelertner and Carriero ’86)

- Coordinables
  - depending on the application context
  - Internet agents, processes in a parallel application, users in a CSCW system

- Coordination media: shared dataspace
  - tuple spaces: bags of tuples
  - tuple: ordered set of typed fields: (int 4, char ‘f’, float 3.14)

Example: Tuple-based coordination models (ii)

- Coordination language:
  - a few simple primitives to put, read, extract tuples:

  ```
  out(int 7, char ‘f’, float 2.78);
  read(int a?, char c?, float 3.14);
  in(int 7, char ‘f’, float)
  ```

- Coordination Laws
  - associative access to tuples: matching of tuples with a provided template tuple (a tuple with formal fields), e.g.:

  the template tuple (int i?, char ‘f’, float 2.78) matches with the tuple (int 7, char ‘f’, float 2.78)

  - synchronization over tuple occurrence in the tuple repositories (an input operation blocks if no match occurs)
Example: Tuple-based Coordination Models (iii)

- **“Classical” tuple-based models**
  - built-in pattern-matching and synchronization mechanisms
  - lack of flexibility

- **Programmable tuple spaces**
  - new behavior of the coordination media (the tuple space)
    can be programmed in response to access events, e.g.,
    - new pattern matching mechanisms
    - new synchronization policies

- **In other words:**
  
  *programmable coordination laws*

---

The Organizational and Social Perspectives

- Coordination models promote an “organizational” or “social” approach to application design and development
  - autonomous components play a role in an organization and their interactions are subject to the rules of the organization in which they are acting
    - coordinables: agents playing a role
    - coordination media: the “workspace” of the organization
    - coordination laws: organizational rules
  - individuals live in a society, have a position in that society, and their social behavior must be coherent to the society’s expectations
    - coordinables: individuals
    - coordination media: a party, a city, a country, ...
    - coordination laws: social conventions
“The Example” of Coordination: Workflow Management

- “Agents” access and modify documents
  - documents reify the concept of coordination
  - workflow rules are actual coordination laws
- Sharp separation between documents and agents

AGENTS (coordinables)

WORKFLOW RULES (coordination laws)

DOCUMENTS (coordination media)

The Case of Document Agents

- Enabling activity in documents adds design choices:
  - How to clearly identify “agents” “media”, and “coordination laws”?
  - documents are agents? documents populate the media to enact coordination laws?

AGENTS (users and external applications)

WORKFLOW RULES (as part of document agents behaviour?)

DOCUMENT AGENTS
XML, Document Agents, and Coordination

- Summarizing:
  - proactive documents, or “document agents” as a suitable paradigm for Web-application development
  - mobility intrinsic in agents and documents
  - coordination models as a suitable framework for agents and document agents applications
  - In addition:
    - XML as the “de facto” standard for Web documents
    - XML naturally promotes the definition of “active documents”

- Starting from that (2nd Part of the Tutorial):
  - building coordinated document agent applications with XML
  - systems and issues

Part 2: XML Technologies for Coordinable Document Agents

- A Taxonomy of Possible Approaches
  - based on the coordination conceptual framework

- XML Coordinable Agents

- XML Coordination Media

- XML Coordination Laws

- Open Issues
Where to Exploit XML in a Coordination Framework?

- **At the level of document agents**
  - document agents directly coordinate with each other
  - the coordination media does not actually exists – it is implicitly formed by all the document agent executing in a context and subject to specific coordination laws

- **At the level of the coordination media**
  - the coordination media as an XML dataspace
  - XML document agents in the data space act in respect of coordination laws associated to them in the dataspace

- **At the level of coordination laws**
  - XML document agents as repository of coordination laws and as active entities enacting them

---

How to Exploit XML: XML Document Agents

- agents and their internal behaviors are represented in XML/XSL + (possibly) supporting code

  - Coordination
    - **subjective** coordination
      - agents’ embedding of coordination laws
      - the coordination media is only a virtual media
      - no explicit representation of coordination laws, and no concepts of “organizations” and “organizational rules”
    - **objective** coordination
      - coordination laws “attached” to agents to make them interact
      - according to explicitly coordination laws

- **Examples:**
  - Displets, XML-Agents
How to Exploit XML: XML Coordination Media

- Document agents of any kind are immersed in an XML Dataspace, and coordinate via it.
- The dataspace exists explicitly and can impose its own coordination laws.
- The activity of document agents is mainly related to their living on the coordination media:
  - XML documents get accessed by external application agents.
  - XML documents are associated to a behavior that establishes how to rule the accesses and the modifications to documents.
- Examples:
  - MARS-X, XMIDDLE

How to Exploit XML: XML Coordination Laws

- Coordination laws are expressed in XML/XSL.
- Documents agents act as the “controllers” of the coordination activities of the applications:
  - XML describing how to react to coordination event.
  - XML+XSL document agents to enact the coordination laws.
- Examples:
  - XML WorkSpaces
### DISPLETS

- Dept. Computer Science, University of Bologna
- **Initial Release:**
  - General-Purpose Visualization Engine for XML documents
- **Latest Releases:**
  - general-purpose framework for active documents
  - and for multi-document-agents applications
- XML document agents ("Displets") can:
  - include re-active behaviors related to rendering, printing, be searched...
  - Include pro-active behaviors to produce animation, generate new documents, search, etc…
  - coordinate with each other
Passive vs. Active Document in DISPLETS

- Seamless representation of passive and active documents
  - XML documents are passive unless
  - an XSL stylesheet is attached to it
- The XSL stylesheet can:
  - transform an XML document
  - produce an active document
  - by attaching Java code to the document or to portions of the document structure

The XML Compiler of DISPLETS

- After parsing of XML document
  - generation of DOM tree
- The DOM tree is transformed accordingly to its XSL stylesheet
  - modified to possibly specify behaviors
- The DOM tree is then transformed into a
  - tree of (runnable) Java objects
- The XML document has become a document agent!
Coordinable DISPLETS

- DISPLET can have associated
  - their own "private" behavior
  - when on a group (when executing in a virtual coordination media) the "coordination" behavior enacting coordination laws
- Static approach
  - compiler-based
  - the document agent (and the associated coordination laws) generate at compile time
  - no dynamic entering leaving of groups

Virtual Coordination Medium

XML Agent

Coordination Laws in XSL

App. Specific XSL behavior

Static approach
  - compiler-based
  - the document agent (and the associated coordination laws) generate at compile time
  - no dynamic entering leaving of groups

A Non-Coordinated System: ADF

- A product of General Magic
- Agent Definition Language in XML
  - XML data as agent attributes
  - XML code as agent behavior
  - very nice, however...
- Agent interactions
  - object-oriented
  - no specification of coordination rules
  - no clear framework
- Not suitable for complex multi-document applications

```xml
<cell name="count">
  <val>3</val>
  <max>10</max>
</cell>

<handler name="inc">
  Increment the value
  <set count.val="\$count.val+1">
    The new value is
    <recall count.val>
  </set>
</handler>
```
MARS-X

- University of Modena and Reggio Emilia, ITALY
- A coordination infrastructure for mobile agents
  - virtual mobility
  - actual agent mobility
  - physical (i.e., device) mobility
- Ported on different agent systems (Aglets, Java2Go, SOMA)
- XML dataspaces for mobile agent coordination
- programmable XML dataspaces

MARS-X: The Architecture

- Local XML dataspaces for mobile agent access
  - shared by a node (or domain of nodes)
  - Linda-like (JavaSpaces) interface to XML data
  - Programmable tuple space model: coordination laws can be programmed in Java
MARS-X: The JavaSpace Interface

- JavaSpaces: the new Linda standard?
- Classical Linda operations with extended functionality
  - `read` (Entry `tmpl`, Tr `txn`, long `lease`)
  - `take` (Entry `tmpl`, Tr `txn`, long `lease`)
  - `write` (Entry `tp`, Tr `txn`, long `timeout`)
- XMARS additions: `readAll`, `takeAll`
- The object tuple `tmpl`:
  - of a class implementing the `Entry` interface
  - its instance variables are the tuple fields
- Object-oriented pattern-matching

MARS-X: The XML Dataspace

- A Document space as a collection of XML documents

```xml
<?xml version="1.0"?>
<!DOCTYPE myEntry SYSTEM "http://mysite/myDTD.dtd">
<infoN>
  <f1>3</f1>
  <f2>foo</f2>
  <f3>blahblah</f3>
  <f4>1?/f4>
</infoN>
<infoN>
  ...
</infoN>
```
MARS-X: Tuple-based Access to the XML Documents

- XML entries as JavaSpaces tuples

```java
class _infoN extends AbstractEntry{
    static final URL DTDfile = new URL("http://mysite/myDTD.dtd");
    public Integer f1;
    public String f2;
    public String f3;
    public Integer f4;
}
_infoN t = new _infoN();
t.f1= null; t.f2 = "foo";
t.f3 = "*bl*"; t.f4 = 17;
myEntry result = space.read
    (t, null, NO_WAIT);
...```

MARS: Coordination Laws

- Meta-tuples to associate reaction to access events (meta-matching)
  \((\text{Ag-Id}, \text{Tuple}, \text{OpType}, \text{ReactObj})\)
- Access the base-level tuple space triggers pattern-matching in the meta-level to look for reactions to execute
- \text{ReactObj}: Java object with a single method (the reaction itself)
- Example
  \((\text{ag@mo.it, null, read, 01})\)
  triggers the reaction method of object 01 when the agent \text{ag@mo.it}
  performs a read operation
MARS-X: Example of a Document in the Dataspace

```xml
<?XML version="1.0"?>
<!DOCTYPE CourseEntry SYSTEM "http://uni.edu/UniCrs.dtd ">
<course>
    <coursename>Computer Networks</coursename>
    <year>4</year>
    <semester>1</semester>
    <teacher>Jane Smith</teacher>
    <lesson>
        <lessonname>Introduction</lessonname>
        <lessonnumber>1</lessonnumber>
        <abstract>blah blah</abstract>
        <reading>
            <author>..</author>
            <title>..</title>
            <book>..</book>
            ...
        </reading>
    </lesson>
</course>
```

MARS-X: Example of an Agent Access to the Dataspace

```java
_lesson foundlesson;
...
_lesson templatelesson = new _lesson();
templatelesson.abstract = "*networks**"
for(i=0; i< sites_of_the_federation.length;i++)
    // for all the sites in the federation
    ( go(sites_of_the_federation[i]);
        // go to the current site in the list
    if ((foundlesson ==
        S.read(templatelesson, null, NO_WAIT)) != null)
        // a lesson containing "network" in the abstract
        go(home);
        // go back home
    }
```
MARS-X: Example of a Coordination Law

- A reaction to log all read access:

```java
class Monitor implements Reactivity {
    public Entry reaction(XMARS s, Entry Fe, Operation Op, Identity Id) {
        SecurityRegister.add("read", Fe, Id);
        // log the access
        return Fe;
    }
    // returns the matching tuple to the invoking agent
}
```

- Installed via the meta-level tuple:
  
  `(MonitorObj, null, read, null)`

XMIDDLE

- University College, London
- A Middleware (i.e., a coordination infrastructure) for mobile and ad-hoc networks
- Handles dynamic connections and disconnections within a network
- The nodes/processes in the network shares a global XML data structure
  
  - The global data structure is an active document
    - it includes protocols for handling data reconciliation and data merging
    - also application specific protocols can be defined
XMIDDLE: The Architecture

- Coordinables are active processes
  - mobile AND/OR executing on a mobile device
- Coordination media as XML tree
  - possibly replicated (even partially) and shared among processes in the reach
- Coordination laws rules modifications to the XML tree

XMIDDLE: Coordination over the XML Tree

- The processes of a network operates and coordinated with each other by manipulating a shared XML tree (i.e., a globally shared document)
- Peer-to-Peer model
- When a process disconnect from the network, it can continue operating on a replica of the tree
**XMIDDLE: reconciliation**

- A disconnected process, can have modified the tree
- OR it can have been modified by other processes
- When reconnecting, the global document tree must be re-constructed in a consistent way
- The document tree integrates reconciliation policies!

![Reconnection of Process A with re-conciliation](image)

---

**Workspaces**

- **CS - Technical University of Berlin**
- **Workflow-oriented system**
  - transformation of XML documents
  - into other XML documents
  - by one or several activities
  - accordingly to specific transformation and synchronization rules (defining the structure of the workflow process)
- **Coordination steps**
  - coordinate the flow of work on documents
  - accordingly to the structure of the workflow process
- **Workflow management as a coordination problem!**
XSL Coordination Rules in Workspaces

- XML Documents are transformed during the execution of the workflow
  - by an "engine agent"
  - which act accordingly to the rule specified in XSL
    - temporal and causal relationship among activities over documents
- XSL is then used to represent "the coordination rules"
  - enacted in the activity of agents over XML documents

Complex coordination rules in Workspaces

- JOIN
  - synchronization over the occurrence of a set of documents

- SPLIT
  - produce multiple documents from a single one
Meta-Coordination Rules in Workspace

- "Workspace coordination language"
  - describe the structure of a workflow process in an XML document
- XSL rules for producing the specific coordination rules
  - an XSL engine transforms the document describing the workflow process into multiple XSL documents
  - each XSL document describe one specific rule to be applied during the workflow process
- Global rules transformed into local, document-specific rules

The XMLSpace Architecture of Workspaces

- A tuple space as a "bag" of XML documents
  - Including XSL rules too
- "Engine Agents" access the tuple space to
  - retrieve matching documents if XSL rules allow
  - and transform them accordingly to XSL rules
Summarizing: The Current Scenario

- Several systems already exploiting XML
  - for document agents
  - for their coordination
- XML exploited at different levels of a coordination model
  - non-uniform
- Little AND/OR non uniform treatment:
  - mobility and openness

<table>
<thead>
<tr>
<th>System</th>
<th>Coordinables</th>
<th>Coordination Media</th>
<th>Coordination Laws</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displays</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARS-X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X-MIDDLE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WorkSpaces</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Our Ideal Scenario

- An “all-XML” development framework
- Built upon the coordination framework
- Uniform XML representation of
  - coordinables: XML data and XML behavior specifications supported by Java
    - mobile, intelligent, and re-configurable document agents
  - coordination media: explicit notion of organizations
    - support for openness
    - support for mobility across and dynamicity of organizations
    - uniform treatment of code, data, and physical mobility
  - coordination laws: re-configurable to the needs of
    - applications AND/OR infrastructures

<table>
<thead>
<tr>
<th>System</th>
<th>Coordinables</th>
<th>Coordination Media</th>
<th>Coordination Laws</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML-IDEAL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Open Issues (1)

- Investigate the relations between work on semi-structured data (as XML is) and mobile computations
  - representing a distributed worlds as an XML
  - mobility as movements in the XML tree
  - coordination as constrained actions on the tree
  - See “Semi-structured computations”, by Luca Cardelli
- Investigate current software approaches in software and physical mobility
  - need of uniform treatment (models and representations) of all types of mobility
  - data mobility, code mobility, agent mobility, user mobility, device mobility
  - see “Software Engineering and Mobility”, by Picco et al.

Open Issues (2)

- Investigate organizational approaches to software development
  - coordinable document agents as “autonomous” organizational entities
  - exploit research results on organizational structures and organizational behaviors to build effective systems
  - see “Organizational Abstractions for the Analysis and Design of Multi-agent Systems, by Zambonelli et al.
- Investigate “coordinating in the large” approaches and scalability issues for future scenarios
  - multitudes of embedded active computer systems
  - multitudes of “in reach” wireless computing devices
  - See “Proactive Computing”, by D. Tennenhouse
Conclusions

- XML is a suitable technology for describing and implementing Document Agents
  - suits the document-centric Web scenario
  - suits the need of associating behaviours to documents and of moving documents
- XML is a suitable technology to build complex Web applications based coordinable document agents
  - XML can be exploited at any level of a coordination model:
    • coordinables, coordination media, coordination laws
  - Semi-structuring (XML Trees) suits mobility and dynamicity
- Open Issues:
  - need of more uniform models exploiting XML at all levels of the coordination framework
  - more support for dynamicity, openness, mobility, scalability

Selected References

- Introductory to Internet and Wide-area Computing


  http://www.luca.demon.co.uk/Bibliography.html


Selected References

- Introductory to Active Documents

- Technologies for Active Documents:
  Microsoft ActiveX, COM, and DCOM: www.microsoft.com/com
  ToolBook: http://www.asymetrix.com

Selected References

- Applications and Issues of Active Documents
Selected References

- **Introductory to Agents:**
  


- **Introductory to Internet Agents and Mobile Agents:**
  


- **Introductory to Coordination Models and Languages**
  


Selected References

- **On Tuple-based Coordination Models**

- **Coordination of Internet Agents**

- **Coordination and agent-based software engineering**
Selected References

- DISPLETS  
  http://www.cs.unibo.it/projects/displets
  

- The ADF for XML-Agents


---

Selected References

- MARS & MARS-X  
  http://sirio.dsi.unimo.it/ MOON
  

- XMIDDLE  
  http://www.cs.ucl.ac.uk/ssegroup.html
  
Selected References

- **XML-space and Workspaces** [http://www.cs.tu-berlin.de/~tolk](http://www.cs.tu-berlin.de/~tolk)
  
  