Turning Processes into Mature (valuable) Programming Tools

Gustavo Alonso
Computer Science Department
ETH Zürich
alonso@inf.ethz.ch
http://www.iks.inf.ethz.ch

A quote

BPEL is geared towards *programming in the large*, which supports the logic of business processes. These business processes are self-contained applications that use Web services as activities that implement business functions. BPEL does not try to be a general-purpose programming language. Instead, it is assumed that BPEL will be combined with other languages which are used to implement business functions (*programming in the small*)

BPELJ: BPEL for Java technology (IBM, BEA)
Programming business processes

Processes

Megaprogramming (Large-scale Interoperation and Composition): a move from coding as the focus of programming to a focus on composition based on a very-high level (mega-)programming language for software module composition. (Gio Wiederhold)
The reasons for the current success
Past history

Processes, business processes, workflow management, etc., have in the meantime quite a long history behind them

- Paperless office and office automation
- Document management and office flow
- Workflow management
- Business Processes
- We have to be honest: The idea has never made it beyond a very small technical niche – if they made it all
- The concerns and mistrust of 10 years ago are still prevalent

Workflow Management

Workflow Management was a hot topic in the first half of the 90’s

- Advanced Transaction Models
- Process re-engineering
- Workflow tools
- Workflow coalition
- Lots of research activity

The workflow bubble bursts:

- Workflows are software and nobody looked at the complete life-cycle
- Building wrappers was way to costly
- Lack of uniformity (each tool had a different paradigm)
- Black-box approach not adequate
- Eventually became a small addition to closed systems (IBM, BEA, SAP …)
Things are going well

- Business Processes are –after a decade of neglect- back on the front page:
  - several standardization efforts
  - a flood of new tools
  - great interest in the topic
  - lots of new expectations

- Things have become significantly easier:
  - Web services remove the need for wrappers
  - XML standardizes data passing and opens the black box
  - Technology has moved ahead (bandwidth, cluster computing, CPU power, storage space, middleware architectures …)
  - Convergence towards a few platforms (Java, .NET) significantly simplifies the target without having to work on a closed system

The time seems to be now

- Web services based e-commerce
  - emphasis on business processes
  - emphasis on standardization of business processes
  - interest in service composition

- Service Oriented Architectures
  - apparently the perfect match for processes
  - loosely couple architectures
  - message passing simplifies interaction

- Globalization, outsourcing …
We had no part on this

- Almost none of these developments are related or triggered by an interest in processes, we are just in for the ride

- Some of the fundamental problems of workflow management were solved for us not by us:
  - Cost of wrappers = web services
  - No way of dealing with data heterogeneity = XML
  - Human activities = emphasis today is on automated processes
  - Implementation = container technology provides all you need

- Additional uses appeared:
  - Grid and cluster computing for scientific applications

- All this makes the notion of processes much more feasible than 10 years ago

Why we are going to fail … (again)
The sciences of the artificial ...

I use a bicycle to go to work, therefore
I can use a bicycle for interplanetary transportation

To use a bicycle for interplanetary transportation
we need to add more gears and increase the
capacity of the attached water bottle

... and this is what all customers want

Wrong focus ...

- Example of a Mars bicycle:

  **UDDI**: “UDDI is the building block that will enable
  businesses quickly, easily and dynamically to find and
  transact with one another via their preferred applications.”
  (from a Microsoft web page)

- Some of the side effects of the vision behind UDDI are still
  around: semantic web, automatic SLAs, dynamic matchmaking,
  etc., even if it is clear that they suffer from the same limitations as
  the original UDDI vision.

- It is easy to focus on the wrong topics
There is no problem in system design that cannot be solved by adding a level of indirection. There is no performance problem that cannot be solved by removing a level of indirection.

- Processes are a level of indirection—particularly in the execution
- Processes are definitely a performance problem
- Layers of indirection with performance problems are tolerated as long as they give enough added value

- What is the added value of processes?
  - The graphic representation?
  - Faster development?
  - Use by non-specialist?
  - Interoperability (e.g., BPEL)?
The problem with processes

Regardless of how great we think they are, processes are still very far from convincing developers, programmers, and even users that they have clear added value.

A possible way out
Programming Language

Are processes this “very-high level (mega-)programming language for software module composition”? 

☐ YES
☐ NO
☐ NONE OF THE ABOVE

The way out

Processes have to become a high level programming language for software module composition = for SOA!!

➤ because there is nothing else (OO languages are not enough)

➤ because a high level representation is the only way to cope with the extreme complexity of composition of large software modules

➤ because it is actually a much better way to program (but we still need to prove it to the world)
So, why are we not there yet?

- Similar mistakes as with workflow management:
  - No unified programming language (each implementation its own)
  - No support for complete lifecycle (including debugging, evolution, shared development, large scale development, optimizing compilers …)
  - The –ilities count (programmability, reliability, scalability …)
  - Focus on the pretty parts not on what is hard to do (semantics of the language, modularity, structure, data handling, portability, features)
  - World changes fast (REST+RSS instead of Web services, data streams as main data sources instead of documents)

But, what about BPEL?
A few technical discussions

OO Languages for processes?

- One can argue that Java and C# are enough

- Easy to show that this is not true

  - A service is not a class or an object, not even a collection of classes or objects. It does not even make sense to think in those terms. Service composition is not object composition.

  - Web services, SOA, processes, everything that is happening around this space is not Object Oriented

  - Hard to see how any of the basic principles of OO programming (inheritance, encapsulation, design, polymorphism) apply to processes except in a very general, vague sense
On data

- The world is becoming message based
  - the basis for SOA (but not necessarily Web services)
  - loosely coupled (discovered as part of MOM)
  - RESTful approach (somewhat)

- This is not OO and very difficult to do in Java-like languages
  - the data and the operations on the data are clearly separated
  - variables are not messages
  - documents are not programming variables
  - documents cannot be passed as parameters of RPC-like calls
  - text-labeled data types (XML) very different from programming data types

For us to do

- Processes still have a difficult relationship with their data
  - black box approach (ignore the application data)
  - lack of proper data semantics (scopes, visibility, persistence –static vs. volatile, global vs. local variables, namespaces, etc.)
  - Mismatch between traditional programming variables, process data, and application data
  - No instructions/elements for data access

- Let’s take advantage of XML
  - XML is for machines not for humans
  - Let the machine deal with XML
  - At the process level, I only see typed variables with some sort of structure (not the XML message)
  - Use advanced editors to create those variables automatically from a DTD or WSDL
  - Compiler looks at what is being used and automatically generates the code that extracts and populates the variables, the others are discarded
Why we will fail

- Correlation sets in BPEL are evil
  - inherited from queuing systems and MOM
  - terribly complex
  - impossible to scale and use in a multi-user setting without much overhead

- Equivalent to asking every Java programmer to make sure they can uniquely identify each and every RMI call through unique values in their parameters

- Correlation sets can and should be hidden from the process programmer – this is what compilers, stubs, and middleware are for

Why correlation sets?

<table>
<thead>
<tr>
<th>WHY</th>
<th>WHY NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation sets allow the system to match a response to a request by looking at some value(s) in the response and match them to some value(s) in the request.</td>
<td>This can be easily done automatically by a stub</td>
</tr>
<tr>
<td>Allegedly, this is needed because there is no connection established (as in RPC-like applications)</td>
<td>Message identifiers, and other information is needed anyway</td>
</tr>
<tr>
<td>Moreover, this is also needed because Web service standards do not have a standard way of identifying messages (ebXML does, though)</td>
<td>This information can easily be used to correlate messages</td>
</tr>
<tr>
<td>A service interface can easily impose additional information to be used by the system (not by the programmer)</td>
<td>It is a programming nightmare</td>
</tr>
<tr>
<td>It is there only to accommodate MOM systems</td>
<td></td>
</tr>
</tbody>
</table>

©Gustavo Alonso. Dept. of Computer Science. ETH Zürich. BPM, Nancy, France – Sept. 2005
On control flow

- Current processes are:
  - Procedural (RPC-like): do this, then that, then …
  - Pipelines: move a document around through different processing stages
- None of them are enough

- Asynchronous interactions
  - message based
  - event based

- Very difficult to do in Java-like programming languages because the paradigm is RPC:
  - What one often reads about using threads to deal with message passing implementations is terrible (same reason as correlation sets, this is not for the programmer to do)

Why we will fail

- There is no way to program with asynchronous messages/events
  - interrupt driven
  - not for the average programmer
  - not scalable to large applications

- BPEL handlers are a good conceptual step but
  - lead to messy code
  - no support on the language to program the handlers
  - intended for something else but misused
  - semantics are not defined

- Very old example: compensation
  - people have been talking about compensation for two decades
  - we still do not know how to do it
  - the problem is not the concept
  - the problem is the low level implementation of the compensation (not covered/supported by BPEL)
Example: compensation/events

- What to do depends on current state
- Current state is not represented anywhere in the process

We need high level constructs that represent events (message arrives, alarm goes off, events X)
- Use control flow constructs to react to these events

Distributed logic?

- To certain extent, event handlers were intended to be able to pick a message from a queue and then start.
- Distributed logic was tried years ago in MOM
- Message brokers support attaching logic to each queue
- One can then build a workflow by programming each processing stage on a series of queues

- This is a terrible idea as a programming approach
- Can only be done with machine support (it is a good low level implementation)
Coming next …

- Service Oriented architectures - Enterprise Service Bus
  - message based
  - MOM revisited (MOM + web services)
  - complexity still left to the programmer

- This might be where we can make a difference with processes
  - program processes
  - compile to message based, queuing infrastructure
  - hide all the complexities of message passing, messages, message formats, queues, monitoring, etc.

For us to do

- Use a programming paradigm that can be easily understood by developers
  - avoid asynchrony
  - avoid distributed logic
  - provide a global view
  - very limited parallelism

- Provide, at the process level, the equivalent of
  - threads, synchronized threads, shared variables
  - variables that represent relevant events and the state of the process itself
  - let the machinery set and reset those variables as needed
  - only structured flow (not like BPEL)
  - use modularization as we know from programming languages (avoid linking everything together)
Software modules

- All these ideas point in the same direction:
  - Make large software modules the working unit
  - Come up with a language for operating on software modules
- Services as interfaces should represent software modules not objects, classes, methods
- Composition should be among software modules, not across methods or functions (unlike web services today)
- Separate the low level implementation details (XML, message passing, web services) from the high level programming language
- Treat the problem as a programming language problem and solve the whole development cycle

A few more comments on BPEL
On SOAP …

SOAP will be used as a low level communication primitive to implement peer-to-peer wide area messaging systems (in the sense of not having a centralized controller as it happens in conventional message oriented middleware).

On WSDL …

WSDL will be used to describe the interface to the service queue without specifying the content of the messages in the queue.
On XML ...

With SOAP used for implementing queues and WSDL describing the access point (but not the service), XML will become the syntax of the plumbing, not the syntax of the applications.

High-level is the key

- BPEL has many good things:
  - declare variables and “import” list
  - blocks and scopes
  - separate specification from implementation
  - important programming constructs considered

- BPEL is too low level
  - XML is not fit for human consumption
  - assumes all data is defined in WSDL
  - not everything is XML
  - correlation sets
  - lacks many programming constructs (BPEL-J)
  - too oriented towards web services
  - too worried about not being portable
Graphics

- BPEL is also too worried about graphical representation

- Graphics are great but only to a point
  - graphical composition should still be supported …
  - … but extended with textual programming for details

- Graphics is what has made many process tools forget the basics
  - variable declarations and type assignments
  - import/export lists
  - modularity
  - Comments !!
Take home message

- We need ONE process programming language
- We need to focus on the value of such a language
  - high-level
  - shorten development cycle
  - make development easier
- We need to start making processes far more efficient
- We need to make processes independent of low level details (XML, web services, messages, SOA, etc.)

... because of wrong focus

“Although BPEL4WS is not such a bad proposal by itself, it is remarkable how much attention this standard receives while the more fundamental issues and problems … do not get the attention they deserve”.

W.M.P. van der Aalst