EXTRACT of the Overview Section

The Compositional Architecture of Distributed Systems

Robert J. DuWors

Connected Systems Group

January 22nd, 2006

A New Way of Looking at Distributed Systems

Foreword

Compositional Architecture is a unifying theory. We currently suffer from the onslaught of divergent architectures: Object Oriented Architecture, Component Oriented Architecture, Data Oriented Architecture, AJAX Oriented Architecture, and Service Oriented Architecture. With so many "Orientations", no wonder that we rapidly become disoriented! None of these can explain the way the web works today, every day. Compositional Architecture looks for what underlies all of these to fit them into a bigger picture, like the pieces of a jigsaw puzzle.

Compositional Architecture looks beyond them too. Compositional Architecture can explain what goes on by loading a simple web page. Compositional Architecture can unite AJAX clients with the requisite highly personalized security model in a distributed environment. Compositional Architecture thus provides distributed system patterns to structure the backend (network) servers accordingly. This unification allows AJAX clients to seen clearly as User Agents that provide yet another kind of network service. Compositional Architect also deals well with the manipulation of Business Processes and Service Orchestration.

And that's just the opening act for Compositional Architecture as found in this Technical Report.

The full version of this report resides at:

http://www.csgroup.com/files/CompositionalArchitectureTechReport.doc

Contents

Figures

Note: The Appendices can standalone from the main report. In particular, *Appendix A: Thoughts on Service Oriented Architecture as a Major Piece of the Puzzle* extends the Protocols Section. Even a cursory review of this paper should include *Appendix A* which may also be used as an introduction to the main body.

The Compositional Architecture of Distributed Systems

Robert J. DuWors
Connected Systems Group
January 22nd, 2006

"The purpose of Architecture is the answering of Application Domain Wants with IT Solutions." [1]

"The function of IT, pure and simple, is the automation of knowledge." [2]

Abstract

This paper introduces the foundations of Compositional Architecture targeted at the understanding and design of distributed systems. Compositional Architecture provides a framework for automating distributed knowledge. This paper treats composition as the key operation to support this goal and introduces the notion of Metabinding coupled with Transclusion. Three major themes as decribed in this report define Compositional Architecture: The Six Fundamental Elements from which all resources are constructed, The Six Fundamental Relationship that apply to all distributed systems, and The Six Degrees of Correctness. Consideration is given to Compositional Architecture as a general model of computing with "fractal-like" self-describing similarity at all levels of abstraction. Compositional Architecture is seen to subsume and surpass Object Oriented (OOA), Component Oriented (COA), and Service Oriented Architectures (SOA). Finally, Compositional Architecture signals a significant shift in established practices: Compositional Architecture is a way of thinking.

Overview of Compositional Architecture

Compositional Architecture is knowledge oriented. Compositional Architecture views software as a media for the acquisition, storage, modification, and active use of knowledge. By their very nature, distributed systems place knowledge content throughout interconnected networks. Naturally, the central issue in distributed systems architecture is the use of this composite knowledge. Functionality results when the right computations come together with the right persistence (memory) at the right time in the right places resulting in the right outcomes reaching the right destinations[3]. Composition is the essence of this gathering together and spreading out of distributed knowledge content.

Compositional Architecture describes distributed systems in which the basic entities are dynamically composed and recomposed out of elemental network resources as needed on the fly. All entities within the network communicate with each other by means of appropriately defined protocols for the retrieval, transfer, loading, and distribution of network resources. The meaningful composition of these far flung network resources expresses the dynamic knowledge content to be used as needed in order to produce the appropriate results

Dynamic resources in Compositional Architecture are composed of four elemental types[4]:

Units of Computation	(uC)	e.g. programs, processes, scripts, rules, DNA[5]
Units of Units of Persistence	(uP)	e.g. data, memory, DLLs, archives
Units of Distribution	(uD)	e.g. messages, parameters, mRNA
Units of Transduction	(uT)	e.g. I/O, sensors, User Interfaces

The distributed system's automated knowledge content is represented by these elemental units singly or jointly. New knowledge arises from computations which generate, transform and compose the fundamental units together into new entities, making for an extremely fluid notion of form and function. To be effective, these units must fully expose all of the knowledge content within them. Any form of "information hiding" is strictly anathema. All knowledge content must be visible so that it may be recomposed in whole or part as needed. The only exception to universal visibility is any knowledge content explicitly hidden at that moment by a security policy[6]. The secured content thus literally drops out of sight and intentionally becomes unusable within that security context.

Since security is yet another piece of automated knowledge, Compositional Architectures are capable of incorporating security policies and rules as parts of themselves. This reflects the ability of Compositional Architectures to be self defining. Security policies are made out of the same four elemental units and subject to the same highly dynamic composition[7]

The basic rules of Compositional Architecture govern how the four elemental types can be assigned (housed) in named locations (nodes) and transmitted across addressable channels (links) in a network to be combined and transmuted. Because units of Computation can flow as easily as units of Persistence, Compositional Architecture uniquely supports "rule flow" as easily as "data flow."[8] Rule flow enables building systems that provide both high performance and highly flexibility – which is otherwise oxymoronic in today's relatively brittle Object, Component and "Service" Oriented Architectures.

Compositional Architecture makes extensive use of three types of binding. The first two, early and late binding, are traditional, whereas the third, meta-binding, distinctively belongs to Compositional Architecture. Early binding pre-defines linkages well before use e.g. binding relational database schemata to instance data. Late binding establishes linkages at the time of use, e.g. program interpreters, object instantiation, polymorphism, and class loader binding. Meta-binding is the third type and it is new. In meta-binding, the entities and the linkages between the entities come into existence through composition of the four elemental types at the time of use according to the meta-rules outstanding at that moment, which themselves may be subject to dynamic change.

The fetching of remote content for local inclusion is known as "transclusion" in web terminology. Transclusion is one of the central concepts in Compositional Architecture[9]. Remotely sourced transclusion plus local content provide the raw material for composition. Leading to the slogan:

"Don't just reach out and touch --- reach out and make it a part of yourself!"

Transclusion is another method of composition in addition to modularity. Modules combine only through their interfaces. This describes software modularization and componentization. It appears superficially to completely describe hardware: the ICs on a board communicate only through their interfaces. While this is true of the final product as a hardware process, it is manifestly untrue of the design process that created the rules for making the hardware components in the first place. Hardware designers use large collection of "cells" that they cut and paste into the final design. In fact, "cut and paste" nicely describes transclusion.

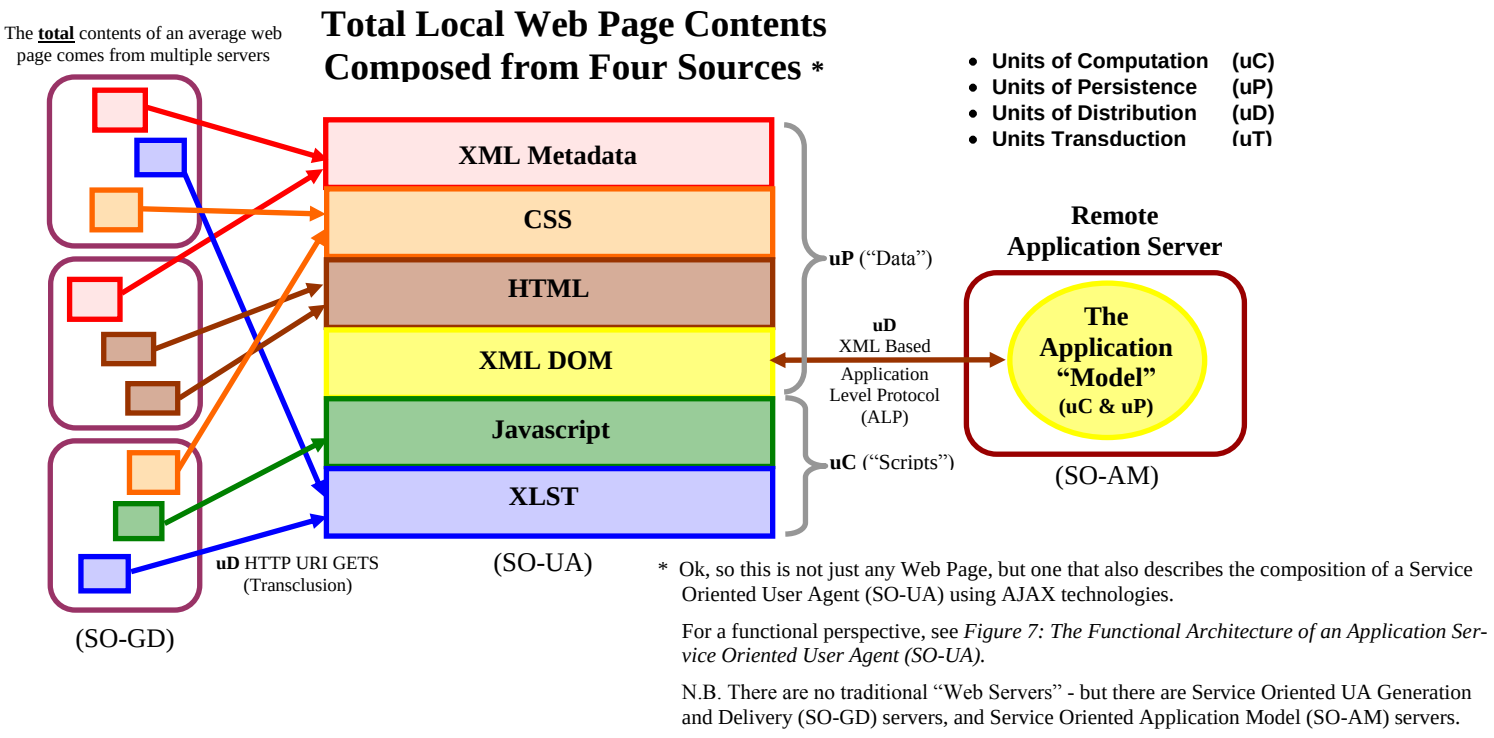
Anyone who has ever used a macro language, a template, a text editor, a mailing list, a Power Point presentation, or a word processor already possesses intimate experience of tranclusion. At the programming level, rather than that of the final software, the whole effort would collapse without "cut and paste[10]".

In the case of Internet technologies, Uniform Resource Indicators (URIs) lubricate the ability to reach out to network resources by means of the appropriate references and protocols. URIs allow resources to be retrieved and redistributed as needed. For example, the following is a partial list of HTML tag types that use URIs to dynamically compose an ordinary web page:

Anchor	Embed	Form	Input	Link
Meta	Body	Frame	iframe	frameset
script	Object	Head	applet	Doctype

Even a simple web page is, of course, made out of multiple units of Persistence and units of Computation delivered on demand from multiple locations across the Internet by units of Distribution and shown to the user by units of Transduction rendered via the browser's User Interface. Thus, Compositional Architecture in comparison to OO, Component, and pure Hypertext models, offers a much better architectural perspective to explain what we actually do and what actually happens every day on the Internet. Compositional Architecture subsumes these other architectural models as special cases and thus surpasses rather than eliminates them. Similarly Compositional architecture subsumes Service Oriented Architecture (SOA) under the more generalized banner of Protocol Centric Architecture (PCA). But is PCA is not just a special case, Protocol Centric Architecture is an integral part of Compositional Architecture, forming its beating heart.[11]

User Agents are the peers of network computing services. User Agents represent the user to the network and the network to the user. The network services may concentrate the resulting processing in one place or spread it out widely, and decide if traces of it should be left behind. One User Agent may use the same service from any number of interchangeable back end servers. Conversely, a server may go looking for any number of User Agents to represent its services to users, i.e. the User Agent IS a network service, one the mediates between the end user and the rest of the network services. While "clients" in client-server architectectures are inevitably 1:1 with servers, User Agents are n:m with services. AJAX allows User Agents to be delivered on demand when needed.



* Ok, so this is not just any Web Page, but one that also describes the composition of a Service Oriented User Agent (SO-UA) using AJAX technologies.

For a functional perspective, see *Figure 7: The Functional Architecture of an Application Service Oriented User Agent (SO-UA).*

N.B. There are no traditional "Web Servers" - but there are Service Oriented UA Generation and Delivery (SO-GD) servers, and Service Oriented Application Model (SO-AM) servers.

Figure 1: Sources of a Web Page

URIs are to Internet technologies are as symbol tables are to compilers and loaders, i.e. the primary linkage mechanism that enables dynamic composition. Thus, both types of linkage mechanism fall within the scope of the Compositional Architecture framework as do Application Program Interfaces (APIs), and Application Level Protocols (ALPs). CSS, Javascript, VBScript, XLST, Java and assorted commercial plugins such as Flash make further use of URIs to compose, display, pass parametric data, and interact with the current web page[12]. Special mention should also go to the upcoming importance of XPath and the OASIS eXtensible Resource Identifier (XRI) proposal.

Of course HTTPXMLRequest has single handily stirred up the Asysnchronous Javascript + XML (AJAX) hornet's nest where a dynamically loaded user agent resides in the bowser on demand. HTTPXMLRequest communicates on behalf of the user agent's internal elements that speak the network Application Level Protocol (APL). HTTPXMLRequest has no direct role in the other major aspect of the User Agent that deals with the localized display and user interaction, i.e. the User Interface (UI) functionality. This clear separation of concerns, network application semantics from final rendering and user interaction, has sent shockwaves through the web development community.[13]

At a higher level of Application Architecture, the following diagram shows the use of Compositional Architecture in the rating component of a high performance telecommunication billing system:

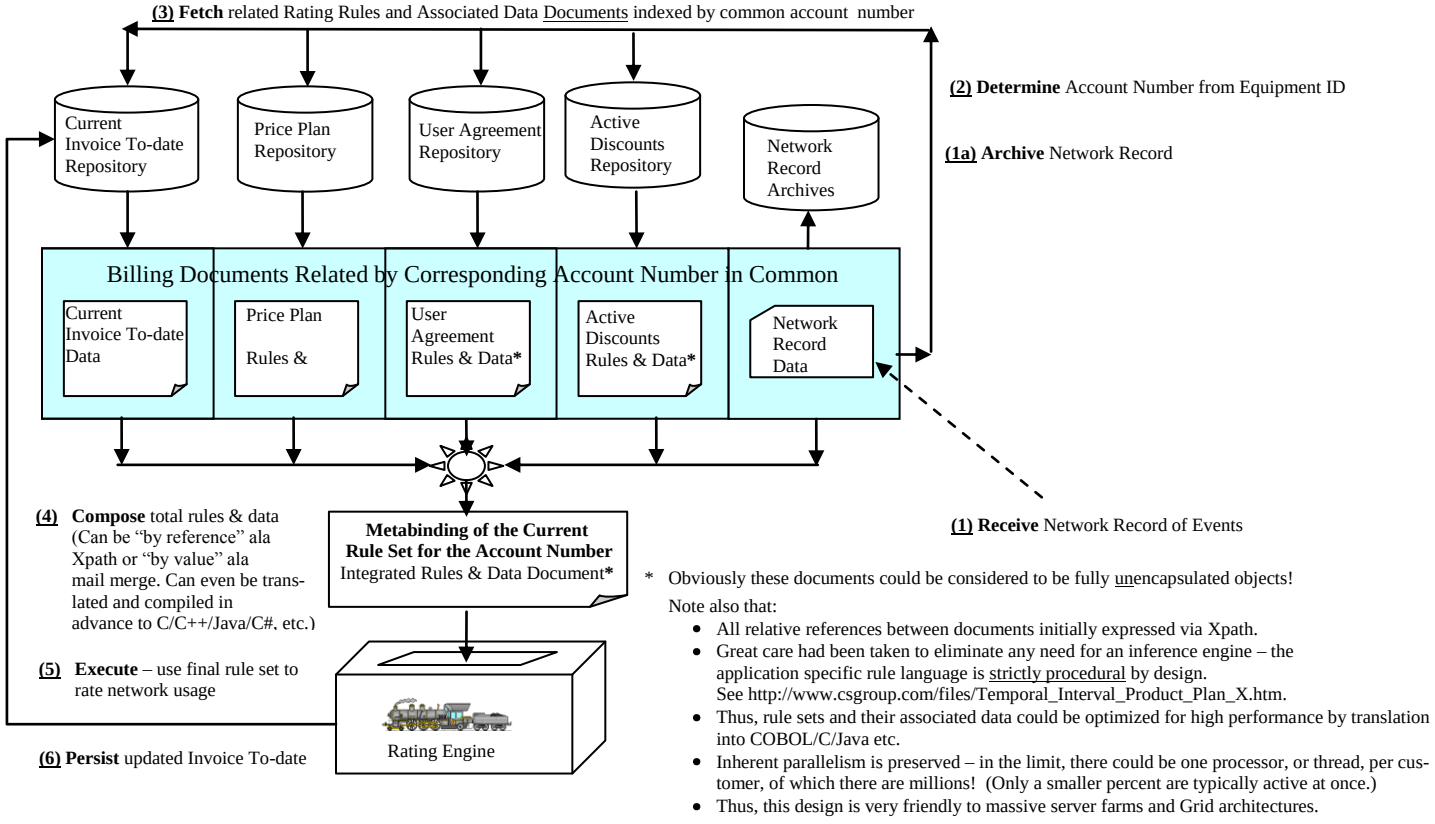


Figure 2: Composition of Business Rules - A Wireless Invoice Rating Example

Because so much dynamic composition is going on, Compositional Architecture often makes extensive use of "meta" relationships such as "describes" or "defines" between layers in the application/system stack as exemplified by the 4 level UML meta-meta layers. It also employs the more traditional "uses" relationship typical of statically layered architectures such as the infamous 7 level OSI model of data communications, and object style interaction.[14]

Employing Compositional Architecture to compose and execute dynamic Business Processes is shown next.

Figure 3: Composition of Business Rules - A Wireless Invoice Rating Example

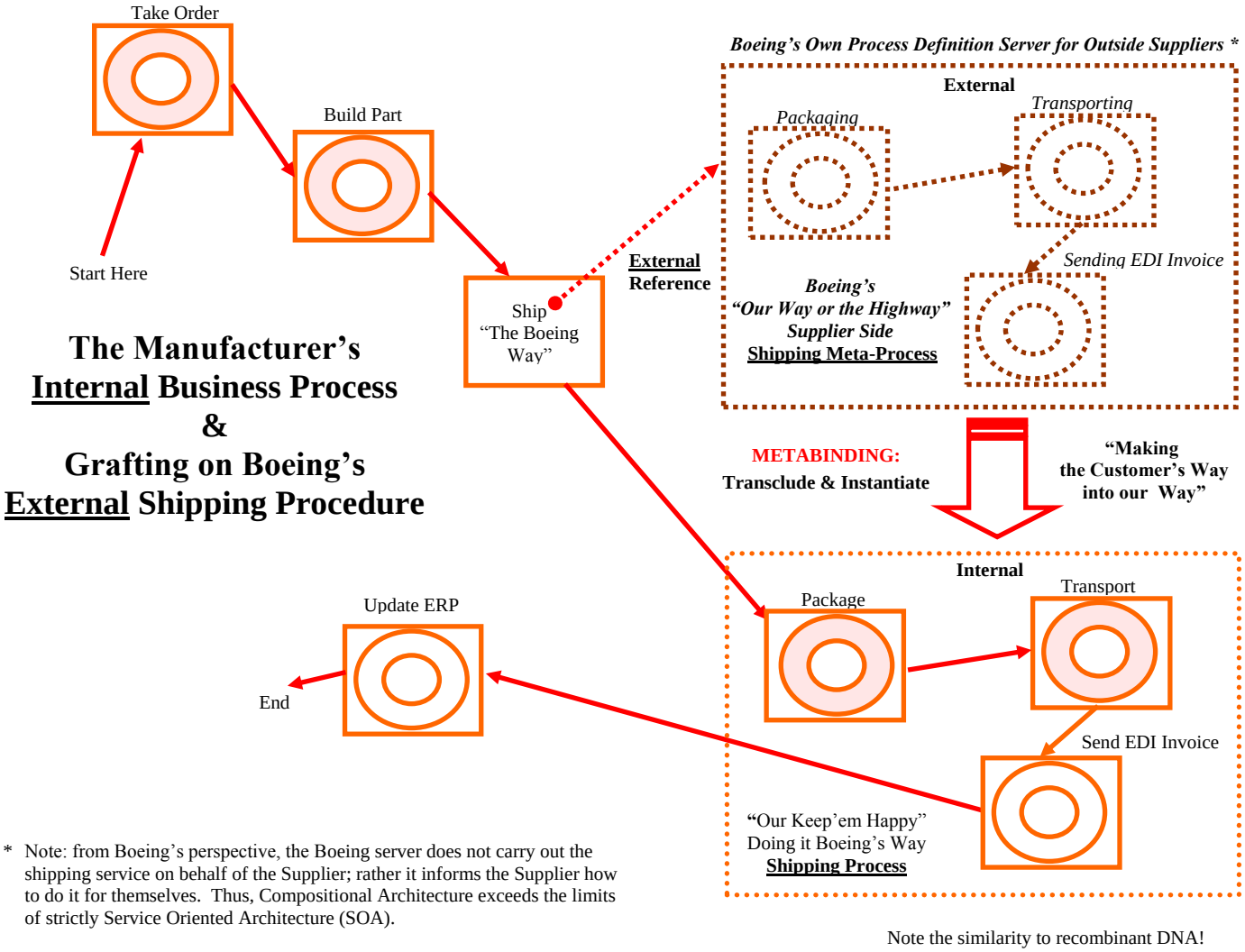


Figure 4: The Dynamic Composition of a Business Process

Much of modern Object Oriented methodology focuses on "patterns". The Gang Of Four (GOF) kicked off this movement with respect to Object Oriented software by defining a collection of low level object patterns. Roger Sessions has used Component Architecture to define a "Software Fortress" model suitable for enterprise class applications.

But - Object and Component models can not be self defining by their fundamental nature, and required the human designer to go outside of them to use other ways of interpreting the application of the pattern. Hence these architectures must generate mountains "design artefacts" and extraneous "architectural models" which ultimately get discarded. However, Compositional Architecture can define these patterns from within, and the linkage from them to the pieces composed and controlled by the pattern's rules of interaction[15]. This is true even if the pattern's rules sets are widely distributed[16]. See *Figure 7: Comparison of Objected Oriented and Compositional Archtitures.*

Uniquely, Compositional Architecture allows for self defining and hence self constructing systems. The very same higher meta-level(s) can serve multiple roles as a design definition, implementation scaffolding, a deployment descriptor, and a run time control. There can be cascading levels where a more generic architectural template creates a specific architectural instance which then brings about the next level of elements in the architecture.

Certainly multi-level definitions and instantiations of this type make a great deal of sense and offer a great deal of power. Because these higher level descriptions can be "constructive" (executable) in the formal sense of the word, they are not wasted. The more this approach permeates downward[17] in system detail, the more flexible the resulting systems. Of even greater merit, this promotes higher levels of knowledge representation throughout the entire system. And thus moves in the direction recommended by Albert Einstein:

"We can not solve our problems with the same level of thinking that created them."

The full version of this report resides at:

http://www.csgroup.com/files/CompositionalArchitectureTechReport.doc

---

[1] Yes, Wants. We don't really need more cars, yet another cell phone, deadlier weapons of mass destruction, or huge shareholder dividends – we simply choose them or not. We can, and in some cases should, do without. So, we don't really have any absolutely necessary "requirements", we only have wants strong enough to act on. Rather than sorting "needs" from "wants", we really need to sort "wants" from "wishes" – only the wants are strong enough to be self justifying calls to action. "Requirements errors" are quite real: a system that meets some arbitrary notions of "needs" and "requirements" but not wants, is worse than doomed – it will generate serious negative repercussions.

[2] This view aligns with the pioneering articles of Phillip Armour. See *The Acquisition and Automation of Knowledge* in the next section. By "knowledge" we mean awareness of the universe, how to manipulate things in the universe, or the same for abstract systems. We will not quibble with epistemological niceties which draw distinctions between "knowledge being only in the mind of an intelligent entity" versus "mere knowledge representation contained in an automated system." In all humility, sooner or later machines may be better at knowledge than we are.

[3] A.k.a. "The Six Degrees of Correctness."

[4] Summarized in ***Error! Reference source not found.***.

[5] As Philip Armour noted, DNA was the first major form of knowledge representation, that of how to build a living organism. DNA is a type of meta-process definition, which describes the production of proteins. DNA demonstrates the composing of computational elements by recombining DNA extracted from two sources (parents) to produce meta-processes containing new life definitions (children). Business processes composed from multiple sources have similar aspects. mRNA shows messages from the DNA archive to build proteins. DNA demonstrates the power of genetic algorithms!

[6] All knowledge content in lieu of any security policy is visible. The security policy may choose to enforce "default permit" (worse) or "default deny" (better). Compositional Architecture in itself does not impose any impediments to the security policy. The rules of the security policy(s) are dynamically composed and applied as needed. The OASIS eXtensible Access Control Markup Language (XACML) employs the dynamic rule set(s) approach, as do most databases which support "views" and permissions. In essence, a computational paradigm should not impose any ot its arbitary restrictions on the security mechanism. Required knowledge content must be made visible as appropriate, i.e. available for transmission (uD), computation (uC) , persistence (uP) and interaction with the outside world (uT). Selective is good, but coyly hidden is not. Thus, Compositional Architecture is very "security friendly." In contrast, Object Orientated and Component architectures tie the hands of the security policy with their own unavoidable "information hiding" that may allow indirect and undetectable "information leaking" contrary to the security semantics of the application.

[7] As shown in *Figure 9: An Example of Security Policy as the Composition of Rule Set***Error! Reference source not found.**. In particular, the security rules applicable to an individual are simply a subset of the total rules of "personalization", namely those personalization rules that restrict functionality and visibility. This observation forms basis of the Distributed Capability Model and its specialized form, the Bifurcated Distributed Capability Model suitable for use with dynamically loading User Agents such as AJAX clients. See *Figure 10: The Traditional Meanings of Roles, Groups, and the Acess Control Matrix* and the preliminary The Distributed Capability Security Model at www.csgroup.com. The XACML standard from OASIS provides a sophisticated model by which to apply the principles of dynamic composition of security rules, although XACML still seems to still be Access Control List (ACL) oriented.

[8] As an example, see Applying Intervals to Actions in a Document Rendezvous Model to Support Billable Event Rating: Beginnings of a Compositional Architecture on the Writings page at www.csgroup.com.

[9] It is reasonable to view a protocols as a series of transclusions, see Protocols in the next section.

[10] Perhaps "copy and paste" more exactly matches transclusion, but "cut and paste" has more cache!

[11] It's debatable whether or not Compositional Architecture subsumes Process Oriented Architecture (POA) in the sense of the Pi Calculus. Despite apparent differences, the fundamental units of uC, uP and uD can map onto "processes." It thus appears that Compositional Architecture can incorporate POA as one of the available formalisms. Protocol definitions, static and dynamic, are one area where POA might excel for rapidly changing Business Processes. Compositional Architecture springs forth from these fertile grounds. It appears that Microsoft has gone in this direction with Windows Workflow Foundation (WWF). WWF does not yet seem to have a tie into SO-UA AJAX clients and their SO-AM and SO-GD back-ends, as that requires a more general framework like Compositional Architecture.

[12] It is well worth noting how multi-linguistic a single web page can be. Furthermore, the computing formalism behind the languages varies radically: most are imperative, but XLST is truly functional and HTML is purely descriptive, some are object oriented and some are not. Both the client and server sides of internet technologies employ an unprecedented bevy of languages at one time. The notion of "knowledge content" is unavoidable in order to bridge them.

[13] This reality initially prompted the effort to codify the architectural principles that have evolved into Compositional Architecture. It was obvious that something new was going on outside of the usual OO and Component Architectures, and Service Oriented Architecture was not complete enough to describe it. However, at that time neither the author nor the developer who had independently discovered AJAX techniques and is also an accomplished architect, could say what. Thus began this journey that has led to many more destinations than originally imagined.

[14] See Figure 5: *Examples of Common Architectural Meta-Models.*

[15] Databases very much fall within the scope of Compositional Architecture, making good exemplars. In theory and practice, Relational Database Management Systems (RDBMS) do a lot more than just store data. They also store a surprising amount of computational ability in the form of triggers and stored procedures. A RDBMs without these would be greatly reduced in value to the point of being competitively crippled. SQL is a very active language for manipulating both meta-level schema and instance data. Without that ability, Databases would face the same problem as first order logic – simple predicates systems can not create themselves and hence require an external means to come into existence. To get off the ground logically, you need "second order" meta-logic, Goedel not withstanding. Meta-levels are powerful, even if they sometimes have surprising consequences.

[16] All too often today's generation of "business rule engines" and "business process orchestration" force the rules to live in only one place at a time. But so not in XACML. XACML gets rule distribution right, or at least some predefined and open ended versions of it.

[17] Or, upwards for that matter. The more it permeates throughout the design and implementation, the better. The literal building of systems by meta-levels in many ways is the ultimate expression of "literate programming" – software that both people and machines can read.