A Resource Oriented Framework for Context-Aware Enterprise Applications

Dave Duggal
Consilience International LLC
dave@ideate.com

William Malyk
Consilience International LLC
bill@ideate.com

ABSTRACT

The Ideate Framework is the result of a property-driven software development effort intended to improve the effectiveness and efficiency of Knowledge-work. The keys to supporting such work are identified as context-awareness and mass-customization, both of which are provided for by the framework in a practical, light-weight, scalable, and adaptable manner. Underpinning the framework is a new hybrid architecture promoting the scalability of distributed enterprise systems and the delivery of server-driven applications. The architecture shares some similarity to the Representational State Transfer (REST) style, against which it is contrasted. In addition, this paper describes the key components of the Ideate Framework, and compares the results against other related approaches.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures – *Data abstraction*, *Domain-specific architectures, Patterns (e.g., client/server, pipeline, blackboard).*

General Terms

Algorithms, Management, Design.

Keywords

Resource-orientation, context-awareness, enterprise, mass-customization, REST, Web Oriented Architecture, Business Process Management, Information Management, Enterprise Resource Planning.

1. INTRODUCTION

Not all Resource-Oriented Architectures follow the specific tenets of the Representational State Transfer (REST) architectural style [9]. REST's constraints are designed to ensure the scalability of Ultra Large Network Systems, and achieve this by shifting the locus of control of a client-server environment from the server to the client minimizing server-side processing.

This work introduces a Resource-Oriented enterprise information system that is intentionally server-centric; leveraging the latent transformability of Resources to provide customized system responses in support of useful automation[1] and required compliance[2] (i.e. business rules; enterprise policies; transaction controls).

The paper begins with background describing the properties required of enterprise resource-orientation. It then provides an overview of the Ideate Framework and contrasts it against REST, then walks through the algorithm on which the framework relies. Finally, other related work is identified and the paper concludes with a discussion of future research.

2. BACKGROUND

The Ideate Framework™ is the result of a property-driven software development effort intended to improve the effectiveness and efficiency of Knowledge-work.

Knowledge-work cannot be sufficiently modeled in flowcharts [6, 7, 10, 11] as it is subject to events and human discretion, making it unpredictable and therefore poorly suited to static process models [2]. Efficiency needs to be balanced with flexibility for business effectiveness requiring dynamic models that support variance. In turn, sustainability requires dynamic models that can likewise adapt without disruption. Therefore, the target implementation needed to provide for mass customization while being practical, light-weight, scalable, and adaptable.

Mass Customization is defined as "producing goods and services to meet individual customer's needs with near mass production efficiency" [17]. The goal of Mass Customization is to improve responsiveness to situational requirements without the delay, expense and limitations of traditional exception and change management, which can result in underserved customers or lost opportunities. It is achieved by "effectively postponing the task of differentiating a product for a specific customer until the latest possible point in the supply network" [5]. Thus Mass Customization is a consistent means for practical management of variance and change.

Responding to situational requirements requires context-awareness [1, 12, 15]. Context-awareness allows run-time evaluation of a system interaction (i.e. client requests and events) so the system can factor a 'big picture view' in calculating a response.

[1] Examples of automation use-cases well suited for Ideate's case management: complex regulatory forms; budgeting-planning-financial management; resource recommendations; etc.

[2] Examples of compliance use-cases well suited for Ideate's case management: financial controls; conflict of interest detection; complex form validation; contra-indications for drugs; resource thresholds; etc.

To realize the target implementation, it was critical that its architecture support underspecified reflective programming. Such programs are generalized so they can be dynamically adjusted, making them suitable for constructing custom system responses based on interaction-context.

The overhead of generating system responses on-the-fly [19] has historically been a barrier to the introduction of such software solutions. 'Dynamic', as commonly used in regards to business process, refers to the ability to chain predefined services based on fixed 'data driven' conditions of the in-bound request. Such process composition supports only a limited and 'expensive' middleware-centric form of variance that does not support adaptation.

The efficiency of recursive run-time transformations is critical to achieving a practical implementation of Mass Customization. To that end, the system was implemented on a Resource-Oriented architecture because of the direct transformability afforded by Resources; the same property that makes Resources the enabler of client mash-ups.

3. IDEATE FRAMEWORK

3.1 Overview

The conceptual architecture of the Ideate Framework focuses on a system controller running in an intermediary. Imposing a system-controller on a Resource-Oriented environment shifts the locus of control to the server. The system customizes responses and directs clients to required or recommended Resources based on the use of 'out-of-band' context.

While the system does not adhere to the REST style, it does yield a practical, light-weight, scalable, and adaptable Information System that supports mass customization and maximum code re-use.

Ideate is optimized for unpredictable and highly-variable knowledge-work driven by human discretion and events[3]; a domain of work underserved by Business Process Management technology. Moreover, Ideate supports process without the cost, indirection or latency of middleware services associated with conventional approaches.

Ideate evaluates each interaction and constructs a custom response that is 'made-to-order' for the situation. The System calculates the minimum rule set necessary to satisfy internal requirements of the interaction context. In this way, the degree of structure (i.e. control logic) varies for each performance of an operation; allowing Ideate to be as flexible as possible and as procedural as necessary.

[3] Examples of knowledge-work well suited for Ideate's case management: Research & Development; Patient Management; Strategic Planning; Emergency Response; Report Writing; Investigation Work; Insurance Claims; Custom Production Work; and virtually any form of Project-based work which are team-oriented, and where the details, timelines and resources are subject to change making the project a living document versus a fixed plan.

Such targeted responses provide individuals and teams with enhanced decision-support and enables processes to follow situational requirements.

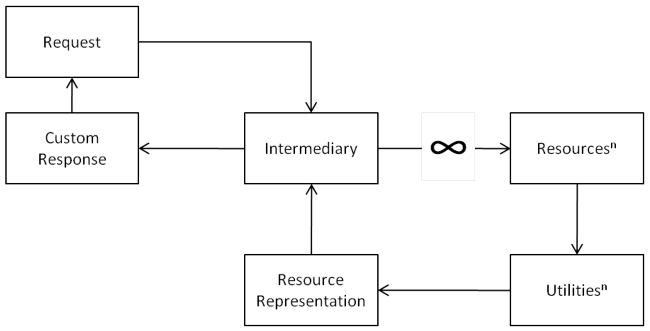

Figure 1. Ideate High-level Components.

3.2 Relationship to REST

Both REST and the Ideate Framework, build on the strengths of Resources. In addition, the REST-style and the Ideate Framework both represent intentional architecture, which is an important subtext of Fielding's seminal dissertation [9].

While the REST style uses its set of constraints for the scalability of Ultra-Large Network Systems and client-driven applications, Ideate uses its set of constraints for the scalability of Distributed Enterprise Systems and promotion of server-driven applications. The two architectures share some constraints, but Ideate is neither a reaction to REST nor an extension thereof; it is simply a Software Framework with its own hybrid architecture. There are several specific reasons why the desired properties of the Ideate Framework are incompatible with the full set of REST constraints:

- Ideate requires an 'understanding' of Resource types and relationships in order to provide mass-customization. In REST's common client-application use-case, the lack of a 'Media Type' standard means there is no current mechanism for 'shared understanding' that doesn't violate other REST tenets thereby limiting capability of REST-based client applications.
- Ideate focuses on context-awareness and customization. In REST, the prohibition against 'out-of-band' context per the State-less Constraint precludes context-awareness (i.e. server-side 'serendipity') and customization of system responses (i.e. server-side programmatic 'mash-ups').
- As an enterprise solution, Ideate requires governance over all system interactions. Client-centricity of State-less Constraint, as well as Identification of Resources and Hypermedia Constraints, restrict server-side (i.e. enterprise) control of interactions.

The violation of select REST constraints sacrifices some scalability relative to REST style architecture (N.B. partially mitigated by design, as well as by increases in Internet bandwidth and server processing capabilities in the last decade). More notably, by shifting locus of control from Client to Server, Ideate abandons client-centricity of REST, including serendipity of client-driven Resource discovery and use, in order to advance the state-of-the-art of enterprise software application design.

Table 1. Relative alignment of the Ideate Resource-Oriented Framework with REST constraints per [9]

Constraint	Support	Notes
Client-Server (5.1.2)	Full	
State-less (5.1.3)	Partial	All state is maintained as Resource state. Application state is not separately persisted by server or client. System uses Out-of-Band context to enrich system response.
Cache-ability (5.1.4)	Partial	Server caching, but no client caching.
Uniform Interface (5.1.5)	Full	
Identification of Resources	Partial	All Resources addressed by URIs. Resources are not directly discoverable, accessible, or linkable by Client.
Manipulation of Resources through Representations	Full	
Self-descriptive messages	Full	
Hypermedia as the engine of application state	Partial	All state is maintained in Resources. Application state is not separately persisted by server or client. Client does not maintain application state.
Layered System (5.1.6)	Full	
Code-On-Demand (5.1.7) optional	Full	

3.3 Technology

The Framework, patents pending [8], provides a flexible architecture with a unified information layer and a common approach to dynamic modeling of process, data, and capabilities. The System is composed of a sole canonical protocol, a single artifact, and one layer – a holistic environment for mashing up information.

All data, business entities, rules, program code, services, encapsulated systems, etc. are stored as Resources that reside in a single layer, a virtual information repository. The Resources are loosely coupled, related by machine generated metadata tags forming a web of dynamic link relations; an information graph akin to a social graph. Resources are indexed for rapid retrieval as with web content. All client and internal system requests use a Uniform Interface.

Every Resource is associated with a 'Utility Type', which identifies the Utility (XML parser; rules engine; workflow engine; reporting engine; indexing engine, etc.) required for processing its representations into a machine executable format.

The System coordinates the processing of Resources to enrich system responses in an Intermediary. Processing is initiated by a System Controller Resource. The System Controller employs an Agent Resource to execute one of a set of Meta-Program Resources that direct all atomic operations of the system.

The Meta-Programs are underspecified reflective programs that define the conditions for their own refinement, requiring run-time interpretation based on interaction-context. In this way, the Meta-Programs enable targeted responses for greater business relevance. Conversely, development-wise, under-specification promotes reuse by enabling situational interpretation.

The Meta-Programs are underspecified in that their conditions are not explicitly defined; they make generalized references to other supporting Resources that must be resolved for each execution.

The Meta-Programs are reflective in that they support generalized references allowing the program to modify its own execution. The system response is always a custom representation of the requested Meta-Program.

Each Meta-Program has three distinct stages: 1) a set of potential generalized actions and the conditions that direct an Agent to perform one or another action; 2) the customizing conditions for each generalized action; and 3) the transition conditions for each generalized action that determines which Meta-Program should be provided along with the response as links for next potential operations.

The links provide dynamically configured control flows that allow a series of atomic operations as part of a greater Activity (i.e. process instance).

The goal of the Agent is to provide a custom system response with valid transitions. Execution is performed through the coordination of other Resources, as directed by the Meta-Program to resolve underspecified conditions, and concludes with a custom enriched representation of the Meta-Program.

The deterministic protocol allows non-deterministic responses (i.e. a consistent method for managing variety). Other probabilistic, semantic or artificial intelligence methods can be included by reference.

$$\begin{pmatrix} (Request\ for\ an\ Operation) \\ +(Meta\text{-}Program\ for\ Operation) \end{pmatrix} \times Interaction\ Context = Custom\ Response$$

Figure 2. Rudimentary Formula of Ideate Construction

3.4 Ideate Algorithm

The Ideate framework responds to each client-interaction using the following algorithm. The components of the algorithm appear in Figure 3.

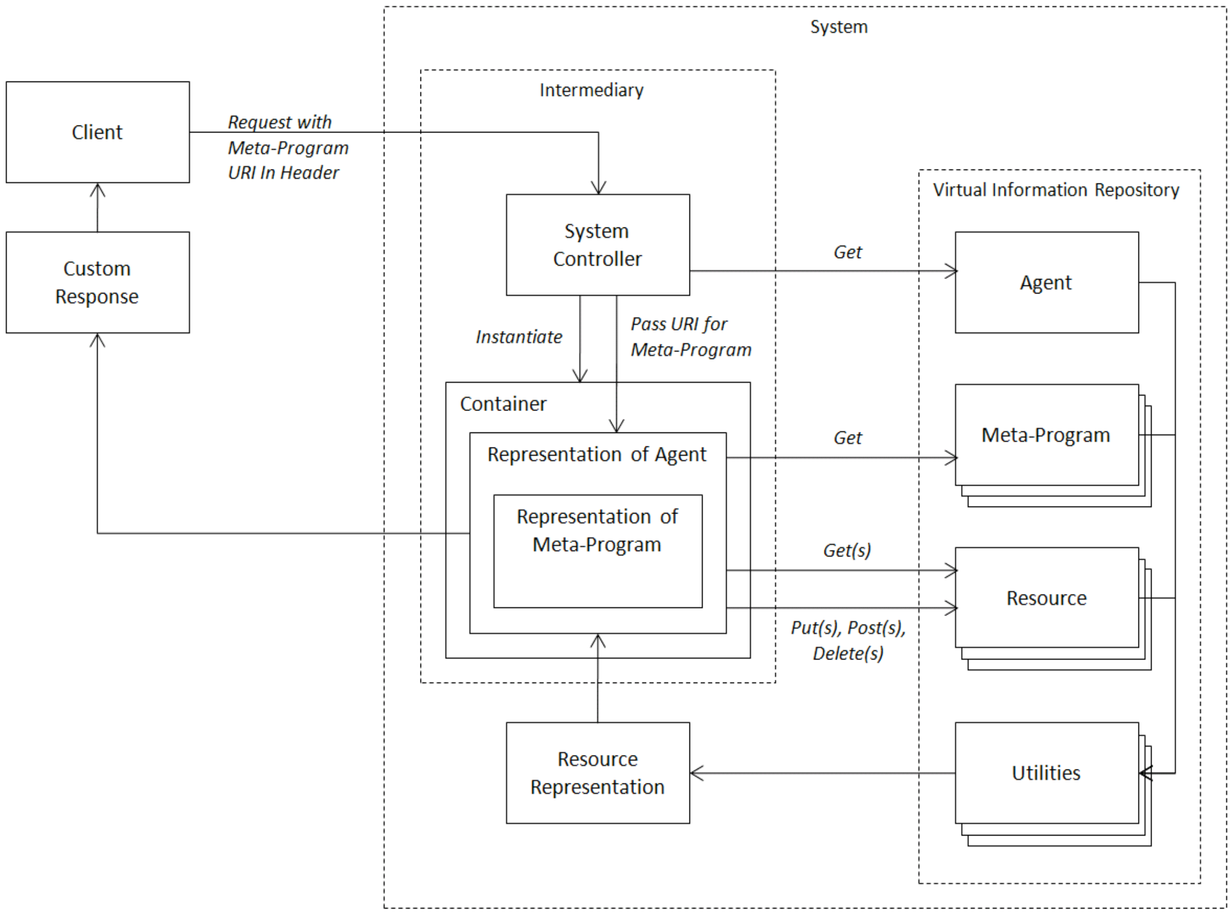

Figure 3. Ideate Architecture as it Relates to the Algoirthm.

1) Client requests represent an atomic operation of the System.
   a. All client requests are addressed to the URI of the System Controller with the URI for the related Meta-Program included in the request header information.
2) A representation of the System Controller Resource initiates pre-processing of the client request on the Intermediary.
   a. The System Controller instantiates a container.
   b. The System Controller requests a representation of the Agent Resource for executing the Meta-Program in the container.
   c. The System Controller provides the representation of the Agent Resource with the Meta-Program URI, and other client request header information (User ID; Activity ID; device form factor; etc.)
3) The representation of the Agent Resource initiates processing of the client request in the container.
   a. The Agent requests a representation of the Meta-Program Resource referenced by the client request for execution in the container.
   b. The Agent uses request header information to fulfill initial Meta-Program conditions as a catalyst for identifying and processing other related Resources.
   c. The Agent 'walks' the link relations requesting and/or transforming Resource representations as directed by the Meta-Program.
      - Each action of the Agent results is an input that resolves one or more Meta-Program conditions, refining the interaction-specific interpretation of the program and advancing its execution (i.e. serial 'trimming' of the information graph).
      - The Agent executes the Meta-Program's three stages sequentially:
        i. Evaluating the program relative to its interaction-context to identify which of the set of potential generalized actions the Agent will perform.
        ii. Evaluating the required generalized action relative to its interaction context to construct a customized system response (user interface update for human client; data delivery for machine client; internal Resource lifecycle maintenance; etc.).

iii. Evaluating the customized response to identify which Meta-Programs should be referenced within the response as valid transitions for next potential operations.

- The goal-oriented Agent performs its service recursively until the program completes with the delivery of a custom system response with valid transitions.

4) Upon completion of the program by the Agent, the System Controller terminates the container releasing all Resource representations.

3.5 Application

At design-time, an application is a virtual cohort of Resources, related by rules, which are available for potential Activity instance of the application. At run-time, an application is the specific performance of application Resources pursuant to an Activity instance.

Instances of Activity are represented by Case Resources. The Case Resource is a virtual folder for organizing Activities and artifacts (documents; forms; people; linked Activities; etc.). The Case inherits emergent properties from its constituent parts. Application state is maintained within Case Resources; application state is not separately persisted on the client or server. The system provides Resource Lifecycle Management for all Resources, with state history and roll-back control.

4. RELATED WORK

From an architectural perspective, Ideate is most closely related to REST; as both share Resource-Orientation, and both are intentional property-driven architectures (see section 3.2). More generally, Ideate has a common bond with all Resource-Oriented solutions, but the designs and intended properties vary greatly as many Resource-Oriented technologies are tools rather than architecture. While Service-Oriented architectures are server-centric and the definition of Service Orientation has been liberalized to include RESTful notions, it is a middleware-centric composition approach, very distinct in design and capability from Ideate. Ideate can interoperate, RESTfully, with Service-Oriented, Resource-Oriented and REST based technologies.

Ideate, with its unified approach to interactions, information and process, is also related to component technologies such as Business Process Management (BPM) Systems (Pegasystems, Lombardi), Event Processing Systems (Progress, Tibco), and Information Management Systems (Informatics, Autonomy). Since Ideate organizes emergent processes in 'Case' folders it also relates to a subset of process technologies called Case Management (Appian, Pegasystems), a niche typified by generic programs for organizing any form of unstructured work. While providing some overlapping capabilities, none of these solutions account for context in an integrated fashion similar to that of Ideate.

As a Framework, Ideate is related to proprietary integrated solutions such as SalesForce.com, Composite Software and InformationBuilders. Capability-wise, Ideate performs a breadth and complexity of functions commonly associated with Enterprise Resource Planning systems provided by Oracle, SAP, Microsoft, Infor, and others.

As a Context Aware Information System (CAIS), Ideate is part of a special class of technologies that seek to increase situational awareness of interaction processing. This is an emerging field, dominated by geo-location systems and recommendation engines. Integration of broad context in process management with run-time customization has historically been considered too 'expensive', in terms of processing power. Leading research in this regard comes from Ploesser et al. [15] who have incorporated an expanded notion of context for adjusting BPM at the strategic level, and from van der Aalst et al. [18] who provide flexible process capable of dynamic adjustment not limited to simple chaining found in many BPM solutions. In comparison to these approaches, the Ideate architecture has a broader scope of adaptation, adjusting all aspects of the resulting system wherein workflow/process is only one part. In addition, alternate approaches to 'fuzzy' conditions such as artificial intelligence and semantic web have been studied extensively, but are still largely theoretical explorations. Ideate's deterministic approach to non-deterministic, emergent, processes represents a breakthrough in this regard.

5. CONCLUSION

The architecture and algorithm described in this paper form the core of the Ideate Framework. The framework has been successfully applied in the knowledge-worker domain to deliver solutions for the research and development (R&D) sector, with future applications planned which will leverage the system's generalized capabilities. Future work will include case studies chronicling these efforts. Experience working with this model has shown development advantages linked to the use of a common unified program model across heterogeneous resource types and efficiencies of reuse, both of which will be the subject of future research.

6. REFERENCES

[1] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M. and Steggles, P. 1999. "Towards a Better Understanding of Context and Context-Awareness", Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, p.304-307, September 27-29, 1999, Karlsruhe, Germany.

[2] Beer, S. 1972. "Brain of the Firm". ISBN: 978-0-471-94839-1.

[3] Box, G. E. P. 1979. "Robustness in the Strategy of Scientific Model Building", Robustness in Statistics: Proceedings of a Workshop. Launer RL, Wilkinson GN, eds. New York: Academic Press; 1979:40.

[4] Chakravorty, S. S. 2010. "Where Process-Improvement Projects Go Wrong". Wall Street Journal, January 25, 2010.

[5] Chase, R B., Jacobs, F. R., Aquilano, N. J. 2006. Operations Management for Competitive Advantage (11th ed.). New York: McGraw-Hill/Irwin.

[6] DeGarmo, T. 2010. "Message from the Editor", TechnologyForecast, 2010, Issue 1. Copyright PricewaterhouseCoopers LLP.

[7] Drucker, P. F. 1966. The Effective Executive.

[8] Duggal, D. and Malyk, W. 2009. Resource Processing Using an Intermediary for Context-Based Customization of Interaction Deliverables. USPTO Patent application and EPO PCT.

[9] Fielding, R.T. 2000. "Architectural Styles and the Design of Network-based Software Architectures", PhD Dissertation.

[10] Flynn, T. P. 2011. "Cutting through complexity", KPMG International Annual Review 2010. Copyright 2011, KPMG International Collaborative.

[11] Hagel III, J., Seely Brown, J. 2010. Designing for Propensity. Blog: http://blogs.hbr.org/bigshift/2010/12/designing-for-propensity.html

[12] Kulkarni, D. and Tripathi, A. R. 2010. "A Framework for Programming Robust Context-Aware Applications", IEEE Trans. Software Eng. 36(2): 184-197 (2010).

[13] Le Clair, C. and Miers, D. 2010. Forrester Update: Dynamic Case Management.

[14] McCoy, D. W. 2010. Context-Enhanced Performance: Reducing Process Stagnation and Chaos. Gartner analyst report, September 20, 2010.

[15] Ploesser, K, Recker, J., and Rosemann, M. 2010, "Supporting Context-Aware Process Design: Learnings from a Design Science Study", 6th International Workshop on Business Process Design (BPD'10).

[16] Taylor, F. W. 1911. The Principles of Scientific Management.

[17] Tseng, M.M., Jiao, J. 2001. Mass Customization, in: Handbook of Industrial Engineering, Technology and Operation Management (3rd ed.). New York, NY: Wiley. ISBN 0-471-33057-4.

[18] van der Aalst, W. M. P., Adams, M., ter Hofstede, A. H. M., Pesic, M. and Schonenberg, H. 2009. "Flexibility As a Service", Proceedings of the 1st International Workshop on Mobile Business Collaboration (MBC'09), volume 5667 of Lecture Notes in Computer Science, pages 319-333, Brisbane, Australia, April 2009. Springer.

[19] Walid, T. 2007. "Resource Aware Programming". GoogleTechTalks, http://www.youtube.com/watch?v=7MIK_ppEXno.