Architecture Centric Tradeoff

A decision support method for COTS selection and life cycle management

Subhankar Sarkar
Senior Manager, Public Sector ERP
IBM USA
ssarkar@us.ibm.com

***Abstract*— Current methods of COTS selection have not been widely accepted in industry, and have been found to lack architectural orientation and a Cost of Ownership perspective. This paper reviews the current methods, and proposes a new method - Architecture Centric Tradeoff (ACT) – for COTS decision support. ACT prescribes a 3-layer Metamodel, Heuristics for Cost of Ownership computations, and a Processor that iterates through candidate solutions to find the optimal tradeoff. In ACT, COTS selection is not driven solely by functional features, but also by architectural characteristics. ACT also takes into account IT portfolio convergence and various COTS delivery methods such as SaaS and Cloud services.**

Keywords- COTS; ERP; Composition based systems; Component evaluation; Cost of Ownership; Tradeoff Analysis

I. INTRODUCTION

Commercial Off the Shelf (COTS) products nowadays comprise a significant proportion of most IT portfolios. In-house software development, following traditional waterfall methodologies, started giving way to *composition based systems* in the late 1990s, and the trend accelerated in the 2000s. Lower costs and shorter implementation cycles were an obvious driver. COTS products provided a viable means to replace outdated systems [1] or integrate disparate portfolios [2]. Also, in the face of the technology revolution, many CEOs were content to leave product development to COTS providers. Around the same time, generally accepted practices and well-formed standards started to emerge in many domains, such as Accounting, Supply Chain and Human Resources. COTS vendors such as SAP, Oracle and PeopleSoft created products in these domains, using design patterns that allowed the same product to be adapted for many businesses. Many organizations adopted COTS as a platform for Business Process Engineering (BPR), and as a means of gaining strategic advantage [3].

Several COTS selection methods exist in literature. One of the first, and the one that gave shape to the generally accepted COTS selection process, was the Off the Shelf Option (OTSO, 1995). This method employed progressive filtering, based on evaluation criteria that included functionality, non-functional properties, strategic considerations and architecture compatibility [4]. Procurement Oriented Requirements Engineering (PORE, 1998), stressed the use of knowledge discovery techniques for progressive elaboration of requirements, and decision support techniques for product ranking [5]. COTS-based Requirements Engineering (CRE, 2002) added a Non-Functional Requirements (NFR) framework to the selection process [6]. COTS-Aware Requirements Engineering (CARE, 2004) intertwined requirements engineering with component evaluation [7]. Mismatch-Handling Aware COTS Selection (MiHOS, 2005) introduced processes for handling mismatches between requirements and COTS, and suggested optimization techniques, such as linear programming [8]. And then, of course, there is the ubiquitous fit/gap spreadsheet.

II. ANALYSIS OF CURRENT METHODS

Most COTS selection methods fit into a general pattern, referred to as General COTS Selection (GCS) [9]:

1. Define evaluation criteria based on stakeholder.
2. Search for candidate COTS.
3. Filter search results based on “must-haves”.
4. Evaluate candidates using decision support techniques.
5. Select COTS, and tailor as needed.

Data suggests that none of these methods have found wide adoption in industry. In a study of Small-and-Medium-Enterprises (SME) in Norway and Italy, it was found that none of them used any of the formal methods for COTS selection [10]. Current criticism for the GCS is summarized below:

- Although most of the proposed approaches were developed for general use, there is no commonly accepted approach for COTS selection [11]. Also, these approaches were proposed without a clear explanation of how they can be adapted to different domains and projects.
- Current approaches suggest using decision making techniques such as weighted score method (WSM) or analytic hierarchy process (AHP) [12]. However, there are several limitations to these techniques [13]. For example, these techniques estimate the fitness of COTS candidates based on ‘one’ total fitness score. This is sometimes misleading due to the fact that high performance in one COTS aspect might hide poor performance in another.
- …what is needed is a more robust negotiation component through which COTS can be progressively selected based on functional and non-functional requirements, architecture, and at the same time resolving conflicts between stakeholders [9].

In this paper, we take a holistic look at the challenges in COTS selection, and discover several problems that have not been adequately addressed in current literature or practice.

 ISBN: 978-1-61208-230-1

Current methods lack a "Cost of Ownership" perspective.

- They look at product features at face value, and select the product with the highest (weighted) feature score. The focus is on the number of mismatches, and on negotiating that to a low value. *The predicate is that the product will not be customized, or that the cost of customization is a function of the number of mismatches alone*. The first predicate is not true in most implementations; usually, the persistent goal in the COTS life cycle is to arrive at the optimal level of customization, not to eliminate customization as a possibility. The second predicate is even less true – the cost of customization is not a function of the number of mismatches, but of the *mismatch type*, and more importantly, of the underlying product architecture and the extensibility mechanisms.
- Development cost is only one component of the customization cost (or the "*Cost of Repair*"), not even the larger part. The life cycle impact of customizations – the potential regressive impact, and resulting increase in the cost of sustenance – is by far the greater cost. That cost, too, is driven not so much by the number of missing features, but by the *mismatch type* and the underlying product architecture.
- Current methods fail to capture the true business impact of accepting a set of mismatches, or the "*Cost of Acceptance*". This cost is not simply the (weighted) mismatch score; it also depends on the level of the requirements hierarchy where those mismatches occur. Mismatches at a higher level, involving foundational requirements, will have a larger cost. The Cost of Acceptance also depends on the mitigation thereof. In the simplest case, the customer organization will stop doing something; then the Cost of Acceptance is simply the value of the lost function. In most cases, the organization will add a manual process, expand another function, or distribute work to another segment of the enterprise.

Current methods lack architectural orientation.

- While many methods mention "architectural reconciliation", there is insufficient detail on how such reconciliation may be pursued. Most of the current methods focus on requirements negotiation, and treat architectural characteristics or non-functional requirements (NFRs), as simply another group of requirements. But architectural characteristics *enable* multiple functions; architectural gaps, unlike functional ones, have a *multiplier* effect on the Cost of Ownership. Current methods do not treat architectural characteristics as enablers, and fail to account for this multiplier effect.
- Current methods do not have a *portfolio perspective*. Architectural characteristics influence IT portfolio convergence, and *ROI* of the organization's IT *portfolio*. For example, if the organization has invested substantially in LDAP services, absorption of a product that does not support LDAP integration will lead to portfolio divergence and diminished ROI. While COTS functionality is best viewed from a Line of Business (LoB) perspective, COTS architecture is best viewed from a portfolio (i.e. CIO) perspective. Then again, if the COTS is delivered as Software as a Service (SaaS) or as a service from a shared community Cloud, the customer organization need not have an equal interest in the underlying architecture, and IT portfolio convergence need not be an issue. Therefore, the COTS *delivery method*, of which there are several in industry today, becomes a factor in the selection process.

III. PROPOSED METHOD

In this paper, we propose a new method for COTS selection and life cycle management – **Architecture Centric Tradeoff (ACT)**. ACT is a decision support method for the entire COTS lifecycle, starting from COTS selection, and persisting through the Design, Build, Deploy and Maintain phases. The fundamentals of the ACT method remain unchanged through the life cycle, while the underlying model data is progressively refined. The salient features of ACT are:

1. ACT explicitly recognizes that COTS based system development is an *optimization, not a construction*, problem. The central object in ACT is the *Tradeoff. Matrix*, not the Requirement Traceability Matrix (RTM).
2. ACT supports a holistic *Cost of Ownership* perspective. In ACT, the Cost of Ownership is a function of the business-product mismatch. The mismatch can be assessed from multiple viewpoints, each resulting in one component of the Cost of Ownership. (Function and Technology are the fundamental viewpoints.) ACT seeks the minima for the Cost of Ownership function, i.e. the collection of Accept and Repair decisions that result in the lowest Cost of Ownership. *In ACT, the COTS product with the lowest minimum Cost of Ownership is selected, which may not necessarily be the product with the highest (weighted) feature score.*
3. ACT is an *architecture-centric process*. It goes beyond features, and explores structural aspects of businesses and products. ACT recognizes the multiplier effect of architectural characteristics such as extensibility. Through the technology viewpoint, the method supports IT portfolio convergence and ROI of IT investments. ACT explicitly recognizes the need for new COTS to leverage IT investments already made.

A. Model Organization and Relationships

ACT comprises of 3 parts:

- **Metamodel**: ACT uses a 3-layer metamodel, constructed in the Unified Modeling Language (UML). The first layer describes the conceptual model, the second the logical, and the third the physical.
- **Heuristics**: These process the model data to calculate the various components of the Cost of Ownership. The heuristics can be adjusted based on organizational assessments and COTS architecture reviews.
- **Processor**: The processor iterates through various candidate solutions, defined by the analyst, to find the optimal tradeoff.

 ISBN: 978-1-61208-230-1

ACT is founded on the *Evolutionary Process for Integrating COTS (EPIC)* [14], which itself is an extension of the Rational Unified Process (RUP). EPIC was developed by the Software Engineering Institute (SEI) at Carnegie Mellon.

- Like RUP, EPIC is *incremental, iterative and architecture-centric*. EPIC uses the well-formed artifacts (e.g. Use Cases) and the modeling language (Unified Modeling Language (UML)) of RUP. But while the constructs are the same, the focus is different. *RUP focuses on the progressive realization of a fixed set of requirements, while EPIC focuses on the systematic tradeoff of requirements and COTS capabilities.*
- EPIC represents a paradigm shift in COTS based system integration. In EPIC, business needs and COTS capabilities converge across multiple iterations. EPIC allows the understanding of requirements and COTS capabilities to evolve along the life cycle. Because it is tradeoff oriented rather than requirements oriented, EPIC reduces risk, decreases cost and facilitates use of delivered capabilities. *Importantly, it also transforms system integration into an optimization problem, which, in the traditional approach, it is not.*

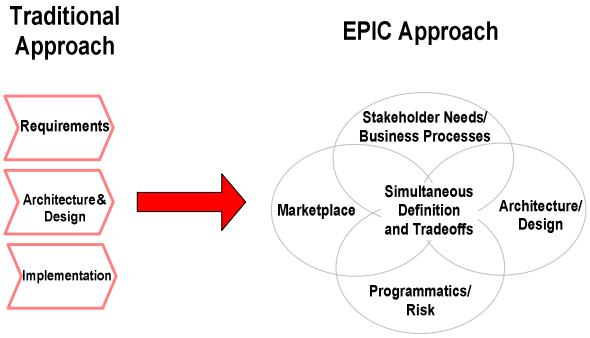

Figure 1. Evolutionary Process for Integrating COTS

ACT builds on the EPIC process framework. ACT quantifies the tradeoffs in EPIC, and facilitates the iterative convergence of function, technology and COTS. EPIC is a broad process framework, and does not say how tradeoffs should be calculated and managed. *ACT takes EPIC from theory to practice; it enables tradeoff oriented COTS program management, governance and tool development.*

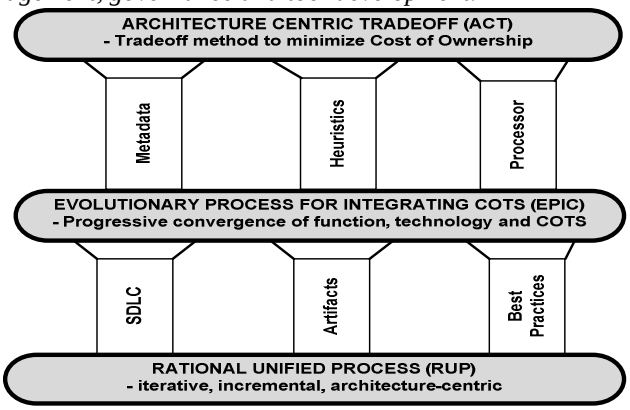

Figure 2. ACT Model Organization

B. *Model Metadata*

ACT is a repository based method. The repository metadata describes the entities in the model and the relationships between them. Inferentially, it defines the boundaries of the model and the subset of problems it can solve. The metadata is in 3 layers - conceptual, logical and physical. The inheritance hierarchy of ACT, as shown the figure below, allows it to work with multiple products, businesses, SDLCs and EA frameworks, while maintaining the same core metadata.

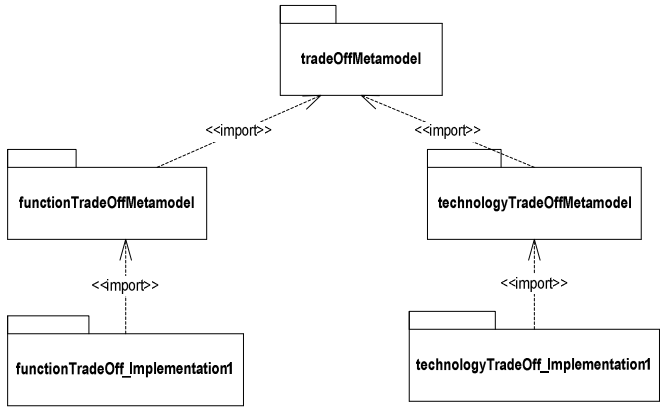

Figure 3. ACT 3-Layer Metamodel

Layer 1 Metadata describes the core concepts.

- "**Enterprise**" is a unit (company, agency, department…) that does, provides or supports "things of value". The enterprise is structured as a hierarchy, with the "relative importance" at each node distributed amongst lower nodes. Function and technology are the two fundamental hierarchies.
- "**Mismatch**" is where the Enterprise is not fully supported by (or does not have) a Product Context. Mismatch can be full or partial. Each mismatch is traced to a specific node in the Enterprise hierarchy, with preference for the lowest possible node, and is fully distributed to "Accept" or "Repair".
- "**Accept**" is where the enterprise needs to do something differently, or stop doing something. "**Cost of Acceptance**" measures the impact to the Enterprise. "Cost of Acceptance" derives from the size and type of the acceptance, the nodes in the Enterprise which it affects, and organizational factors, which may, in turn, inherit from the business domain. Note that a mismatch in one node may have to be resolved by doing things differently at other nodes. This situation is common when consolidating enterprises on a single COTS.
- "**Repair**" is where the product needs to be changed to support the enterprise. "**Cost of Repair**" measures the impact to the life cycle cost of ownership. "Cost of Repair" derives from the size and type of the repair, the nodes in the product context which it affects, and product technology factors, which may, in turn, inherit from the technology domain.

 ISBN: 978-1-61208-230-1

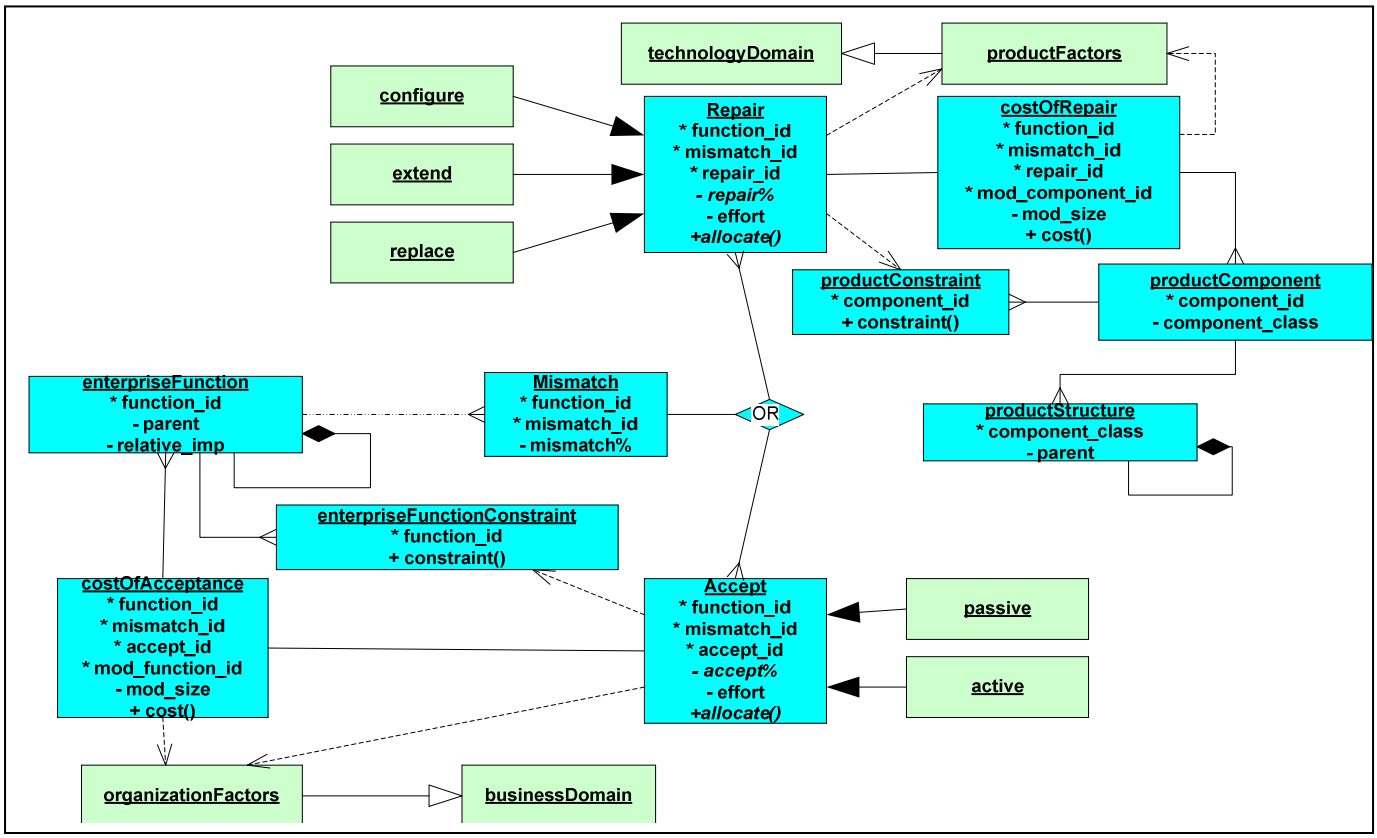

Figure 4. ACT Layer 2 Metadata

Layer 2 Metadata implements the layer 1 concepts as logical constructs.

- "**Enterprise Function**" in Layer 2 implements "Enterprise" from Layer 1. The Enterprise Function hierarchy contains the functional decomposition for the Enterprise. The "relative importance" at each node is a product of the "relative importance"s along the path to that node.
- **Product Structure** is a function-oriented decomposition of the relevant section of the COTS product. (There is no need to model the entire COTS.) Product Component relates specific repair candidates to the Product Structure.
- "**Accept**" is extended into its subtypes – "Active" and "Passive". Passive is where the enterprise stops performing a function e.g. stops selling a product, because the COTS does not support it. Active is where the enterprise reorganizes work, adds manual processes, trains employees or adds compensating controls, to resolve a mismatch. The available subtypes for a function/mismatch depend on the Enterprise Constraints. For example, Passive will not be available for mandatory functions. "**Cost of Acceptance**" is influenced by the type of the Acceptance, and the nodes in the function hierarchy that are impacted by the Acceptance. Cost of Acceptance is also influenced by *organizational factors*.
- "**Repair**" is extended into its subtypes – Configure, Extend and Replace. *Configure* is where only certain literals that drive product behavior (i.e. configuration data or settings) need to be changed. *Extend* is where components may be extended to provide new functionality without modifications to the delivered COTS metadata, such that there is no potential for regressive impact to adjunct components. *Replace* is where there are modifications to the COTS metadata, and thereby potential regressive impact or loss of upgradeability. The subtypes available depend on the *Product Constraints*. Where only certain components are exposed through APIs, for example, the Extend subtype is available only for those components. "**Cost of Repair**" is influenced by the type of the Repair, and the level in the component hierarchy where the changes are taking place. For example, changes at the structural layer (e.g. database schema) will have a greater Cost of Repair than that of changes at the presentation layer (e.g. JSP pages).

Layer 3 describes specific implementations; the logical constructs of layer 2 are implemented for a specific business and candidate COTS. Key activities include formation of the function hierarchy and product structure. In addition to Function, Technology is another fundamental viewpoint. The

 ISBN: 978-1-61208-230-1

technology hierarchy, also known as the *Technology Reference Model*, shows the relevant technology platforms and services within the organization. The relative importance at each node indicates the technology investment at that node. The principle is that ROI targets can be achieved by acquiring COTS that leverage IT investments already made. However, if the COTS is provided as SaaS or Cloud service, Technology ceases to be a viewpoint, and the choice of COTS may be made based on the Function viewpoint alone.

C. Model Heuristics

ACT is a quantitative model for COTS decision support, and the ACT metadata is structured to support computation and optimization. Much of the computations that follow are simple SQL operations on the ACT database. The model has to be calibrated, and the coefficients established, for each organization and COTS implementation program. (The costs shown are notional, and are meant to support relative comparisons, not absolute dollar estimates.)

Cost of Ownership

$$\mathbf{C\_own = C\_accq + {}_{\rho}\sum W_{\rho}*(C\_acpt_{\rho} + C\_repr_{\rho})}$$

- C_own : Cost of ownership
- C_accq : Cost of acquisition (or licensing cost)
- ρ : Viewpoint (Function/Technology)
- W : Weight assigned to viewpoint ρ
- C_acpt : Cost of acceptance (from viewpoint ρ)
- C_repr : Cost of repair (from viewpoint ρ)

Cost of Acceptance

The Cost of Acceptance has passive and active constituents. Passive is where the enterprise stops performing a function because of a mismatch. Active is where the enterprise reorganizes work, adds manual processes, trains employees or adds compensating controls, to resolve a mismatch.

$$\mathbf{C\_acpt = C\_acpt\_passive + C\_acpt\_active}$$

$$\mathbf{C\_acpt\_passive = \prod_i EM\_flex_i *_m \sum F_m * FM_m * MA_m * A_m * AP_m}$$

- EM_flex: Effort multipliers related to the (lack of) organizational flexibility
- m: Mismatch pointer
- F : Relative importance of the node where there's a mismatch
- FM: The percentage of mismatch
- MA: The percentage of the Mismatch that is Accept
- A: The size of the acceptance.
- AP: The percentage of Accept that is Passive.

$$\mathbf{C\_acpt\_active=\prod_i EM\_adapt_i *_m \sum A_m*(1-AP_m)*\{{}_n\sum F\_mod_n * mod\_size_n\}}$$

- EM_adapt: Effort multipliers related to the (lack of) organizational adaptibility
- m: Mismatch pointer
- n: Modified function pointer
- F_mod: Relative importance of the node modified
- mod_size: Size of the modification

Effort Multipliers

Organizational flexibility: These multipliers quantify the constraints that impede an organization from dropping a function in the event of a mismatch. Where these constraints are severe, e.g. where the organization is bound to fulfill certain functions by law, the EM_flex will be high, and Passive acceptance will lead to an unsustainable Cost of Acceptance.

Organizational adaptability: These multipliers quantify constraints that impede an organization from reorganizing work or workforce to resolve a mismatch. Where these constraints are severe, e.g. where there are restrictions on hiring, the EM_adapt will be high.

Cost of Repair

The Cost of Repair comprises of the *Cost of Configuration*, *Cost of Extension* and *Cost of Replacement*. The product architecture, and the technology domain from which it inherits, will place limits on what options are available for a given repair.

$$\mathbf{C\_repr = C\_cfg + C\_extn + C\_repl}$$

$$\mathbf{C\_cfg = \prod_i EM\_cfg_i *_r \sum \{R*RC_r *(1 + {}_n\sum mod\_size_n^{1/\lambda})\}}$$
$$\mathbf{C\_extn=\prod_i EM\_extn_i *_r \sum \{R*RE_r *(1+ {}_n\sum mod\_size_n^{1/\lambda})\}}$$
$$\mathbf{C\_repl=\prod_i EM\_repl_i *_r \sum \{R*RR_r *(1 + \eta *{}_n\sum mod\_size_n^{1/\lambda})\}}$$

- EM_cfg: Effort multipliers related to configurability
- EM_extn: Effort multipliers related to extensibility
- EM_repl: Effort multipliers related to overwrite
- r: Repair pointer
- n: Modified component pointer
- R: Size of the repair
- RC, RE, RR: Percentages of Repair that are Configuration, Extension or Replace
- mod_size: Size of the modification to the component, measured as Source Lines of Code (SLOC), etc.
- λ: The level of the modified component in the hierarchy. More foundational the level (lower λ), higher the effective size of the modification.
- η: The number of upgrades expected during the product life cycle, e.g. for a product life cycle of 10 years, where the vendor releases 2 upgrades a year, η will be 20.

Effort Multipliers

These effort multipliers reflect the product's architectural characteristics.

Configurability:

- (Lack of) Configuration wizards or utilities
- (Lack of) Configurable rules engine and field edits
- (Lack of) Modularization or distinct inter-module interfaces

 ISBN: 978-1-61208-230-1

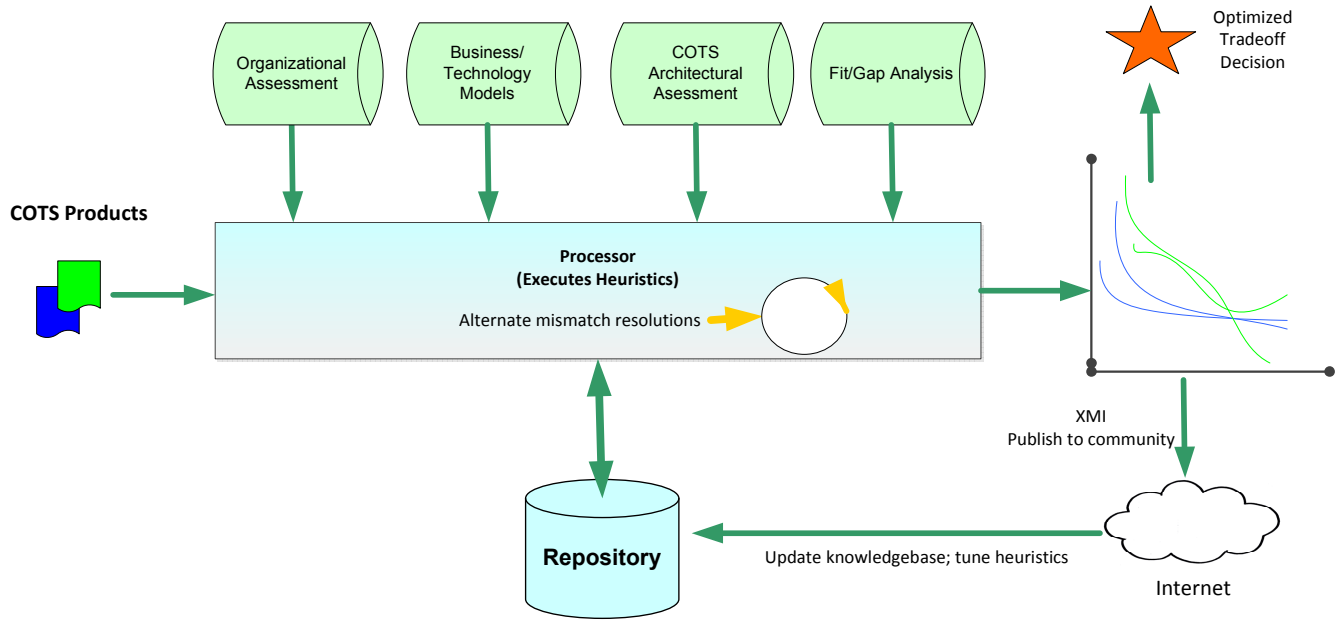

Figure 5. The ACT Processor

Extensibility:

- (Lack of) Integrated Development Environment (IDE), or support for common IDEs (e.g. Eclipse).
- Proprietary programming languages; (Lack of) Support for Java, C++, etc; (Lack of) Object orientation
- (Lack of) Published APIs, user exits and extensible abstract classes

Replacability:

- (Lack of) Integrated Development Environment (IDE), or support for common IDEs (e.g. Eclipse)

D. *Model Processor*

Several iterations take place in the COTS life cycle, some before COTS selection, and some after. The processing sequence for a given iteration during the COTS evaluation phase is as follows.

1. The metadata is populated with organizational assessment results, business and technology hierarchies, and model scaling factors and coefficients.
2. COTS products are fed into the processor. For each COTS, the metadata is populated with results of fit/gap and architectural assessments.
3. The Processor is run with alternate sets of mismatch resolutions, and Cost of Acceptance and Cost of Repair is calculated for each set. The Processor outputs performance profiles for each COTS, which graphs the Cost of Ownership against the level of repair.
4. Finally, an Optimized Tradeoff Decision is reached, i.e. a COTS product at *a specific repair level* is selected.

Figure 6 shows 2 COTS – B has a better feature match, while A is more extensible. The minima of the Cost of Ownership function is lower for A than for B. Most current COTS methods will select B; ACT will select A.

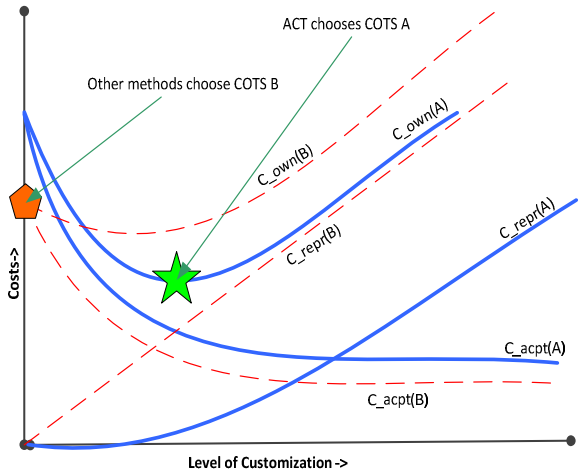

Figure 6. Example of ACT Processor output

IV. CONCLUSION AND FUTURE WORK

ACT provides a comprehensive framework for COTS selection, implementation and governance.

Benefits to the customer organization include:

- Enables a true Cost of Ownership perspective.
- Enables COTS selection at optimal customization, rather than as delivered.
- Costs of Acceptance and Repair are adjusted to organization and product.
- Emphasizes architectural capabilities rather than features; exposes what matters in the long run.
- Explicit shift from requirements to tradeoff collapses project schedules, cost and risk.
- Enables IT portfolio convergence and ROI.

 ISBN: 978-1-61208-230-1

Benefits to the system integrator include:

- Standard analytical model for COTS/business convergence reduces project risk.
- Quantitative model facilitates client communications and Business Process Engineering (BPR) negotiation.
- Common metadata and process model can be reused across engagements. Continual refinement of models, based on engagement feedback.

Future work in this area will include the formation of an adjunct framework to assess configurability and extensibility of COTS products. Future work will also include establishing UML profiles for the metamodel, and finally, the calibration of the model over multiple iterations in the field.

V. REFERENCES

[1] R. O'Callaghan,*"Technology Diffusion and Organizational Transformation: An Integrative Framework"*, Idea Group Publishing, 1998.

[2] M.L. Markus, *"Paradigm Shifts - E-Business and Business/Systems Integration"*, Communications of the AIS, 4 (10), 2000.

[3] T.H. Davenport, *"Mission Critical: Realizing the Promise of Enterprise Systems"*, Harvard Business School Press, 2000.

[4] J. Kontio, *"OTSO: A Systematic Process for Reusable Software Component Selection"*, Univ. of Maryland report CS-TR-3478, 1995.

[5] C. Ncube and N. A. Maiden, *"PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm"*, Second International Workshop on Component-Based Software Engineering, Los Angeles, 1999.

[6] C. Alves and J. Castro, *"CRE: a systematic method for COTS components Selection"*, XV Brazilian Symposium on Software Engineering (SBES), Rio de Janeiro, 2001.

[7] L. Chung and K. Cooper, *"Defining Goals in a COTS-Aware Requirements Engineering Approach"*, Systems Engineering, 7(1), Wiley, 2004.

[8] A. Mohamed, G. Ruhe and A. Eberlein, *"Decision Support for Handling Mismatches between COTS Products and System Requirements"*, ICCBSS'07, Banff, 2007.

[9] A. Mohamed, G. Ruhe and A. Eberlein, *"COTS Selection: Past, Present, and Future"*, *ECBS'07,*Tucson, Arizona, 2007.

[10] M. Torchiano and M. Morisio, *"Overlooked Aspects of COTS-Based Development"*, IEEE Software, 2004.

[11] G. Ruhe, *"Intelligent Support for Selection of COTS Products"*, LNCS, Springer, vol. 2593, 2003.

[12] J. Kontio, G. Caldiera, and V. R. Basili, *"Defining factors, goals and criteria for reusable component evaluation", CASCON'96,* Toronto, 1996.

[13] C. Ncube and J. C. Dean, *"The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components"*, *ICCBSS*, 2002.

[14] C. Albert and L. Brownsword, *"Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview"*, (CMU/SEI-2002-TR-009), Software Engineering Institute, Carnegie Mellon University, 2002.

 ISBN: 978-1-61208-230-1