

Matching Requirements with Off-the-shelf Components at the Architectural Level

Matthias Galster
*Department of Electrical and
Computer Engineering
University of Calgary
Calgary, AB, Canada
mgalster@ucalgary.ca*

Armin Eberlein
*Department of Computer
Engineering
American University of Sharjah
Sharjah, United Arab Emirates
eberlein@ucalgary.ca*

Mahmood Moussavi
*Department of Electrical and
Computer Engineering
University of Calgary
Calgary, AB, Canada
moussam@ucalgary.ca*

Abstract

Off-the-shelf (OTS) components and OTS-based development (OBD) significantly impact software development practices and product quality. In the early stages of development, software architectures are often built by creating new and / or combining existing components. Nevertheless, we still lack common frameworks for OBD for these activities. This paper presents an approach to integrate software OTS components when making the transition from software requirements to architectures. The approach is based on a selection technique using architecture-relevant decision criteria and performing an optimization which is iteratively refined by human input.

1. Introduction

Software architectures play an important role for achieving requirements and quality goals. Architectures describe the primary structure of a software system. In this context *components* play a crucial role as structural artifacts. Additionally, components are first-class candidates for *reuse*. OBD, which is based on reusable components, has become a popular practice when developing software intensive systems.

OBD affects business, management, strategic, legal and technical issues [3]. It impacts design, testing, maintenance, project management, cost estimates, etc. In this paper, we present an approach for partially translating requirements into architectural artifacts by using OTS components. The approach is based on a selection technique which focuses on essential architectural concerns from a functional perspective. Due to the trade-offs between selection alternatives, we emphasize that a selection mechanism can unlikely

provide *the* optimal solution but instead offers decision support for choosing alternatives.

We believe that OTS selection can be done in two different ways: Traditional approaches gather OTS product information, analyze vendors, etc. at the time of selection. The benefit is that only those OTS are analyzed which are relevant for the problem at hand. The other way does analysis of prospective OTS and their vendors up-front and stores them in a repository for later use. The actual selection process is performed later, based on this repository. Here, the actual selection process requires less effort. On the other hand the analysis process likely requires more effort because the number of OTS is higher as many different non-problem specific OTS are taken into consideration. This works best on an organizational level as the additional initial effort is amortized across multiple projects. We will follow the second way.

The benefit of the proposed approach is fourfold: 1) (OTS-) component selection is done based on an OTS repository rather than gathering information from many different sources (i.e., no need for market analysis, vendor selection, contract management, etc. at the time of OTS selection and integration). 2) The focus lies on architecture-relevant selection criteria. 3) We explicitly consider different abstraction levels of OTS and their relation to architectural perspectives in terms of views. 4) The result is a set of multiple OTS covering a broad range of functionality instead of one single OTS most suitable for one specific functionality.

The second section of this paper provides a more detailed problem description. In section 3 we present related work. Section 4 introduces fundamentals important for this research. Section 5 describes the approach which will be illustrated and evaluated in section 6. Conclusions will be drawn in section 7.

2. Problem description

Software architectures are the foundation for all subsequent activities of the development process. Not having an architecture or using poor architectures is a major risk for software projects [6]. On the other hand, constructing architectures is a complex activity. A large number of requirements, stakeholders, and organizational issues must be integrated. OTS, which allow reuse of existing artifacts, are one way to reduce this complexity. But reuse brings other difficulties, e.g., selection, integration, or security issues. Moreover, we might end up with heterogeneous architectures with highly diverse components. Another challenge is that architectures and OBD impact each other: Architectures constrain reuse by global restrictions, like security issues, whereas the selection of OTS influences the architecture [4]. Additionally, researchers even cannot agree on the definition of OTS components [8]. Considering this, it is essential that we develop clear terminologies and process models for the early inclusion of OTS.

In particular, the presented research is motivated by: 1) the lack of reusable OBD methodologies, and 2) the gap between requirements and architectures [2, 7].

3. Related work

Several methods for OTS selection have been proposed, often focusing on the selection of a single commercial OTS (COTS). Additionally, most of the previous approaches provide only a high-level process description and neglect architectural details.

Multiple COTS selection is supported by EPIC [1]. It is a heavy-weight process for COTS-based development and covers a wide range of activities. EPIC proposes 3 views that can be integrated in the Rational Unified Process (RUP) [6]. EPIC gives no details on how to actually do component selection.

Vigder et al. present an approach to evaluate architectural properties for systems built with COTS [11]. It focuses more on the evaluation of these architectures rather than the actual selection of OTS.

Cooper et al. discuss COTS-aware architecting, which heavily relates Requirements Engineering (RE) and architecting [3]. Again, this approach addresses COTS specific aspects in OBD.

PORE focuses on the selection of COTS during RE [9] but does not address architectural concerns.

OTSO provides a generic process for multiple OTS selection [4] that can act as a basis for a specific selection process. It describes phases for a selection process: Our work focuses on the phases of screening, evaluation, analysis and deployment of components.

The pre-selection paradigm provided by OTSO is similar to our approach assuming an up-front OTS evaluation when adding OTS to the repository. It reduces the number of OTS alternatives that need to be analyzed in detail.

4. Fundamentals

4.1. Components

We will refer to OTS components as structural entities of combined functionality obtainable off-the-shelf. Moreover, we define OTS as high-level components. This is because the presented approach focuses on architecting. Architecting stops at major elements that have pervasive and long-lasting effects on the qualities of the system [6].

In further detail, we define OTS as reusable subsystems with high internal cohesion and low external coupling that fulfill a large part of business-level requirements. Additionally, OTS also should meet the following attributes: 1) can be used “as is”, 2) support custom code (wrapper or “glue”), 3) are tailorable (partial change of internals), 4) are interoperable with other architectural entities, and 5) are well-documented.

OTS usually follow unique architecture assumptions rather than universal or consistent architecture paradigms. That means that different OTS use different architectural patterns or styles. This could cause unexpected and unpredictable behavior with other artifacts in the architecture.

Within the scope of this paper, we do not differentiate between types of OTS (e.g., Open Source Software, COTS, Government OTS, Niche OTS, Modifiable OTS, Non-development items).

4.2. The role of OTS across development activities and in architectural views

Software OTS can be deployed at different development phases, such as architecting, design, or implementation. Taking this into consideration OTS could exist as source code, executables, design components, test cases, documentation, etc. [5]. Therefore, we specify 3 types of OTS:

T1. Executable (can run as stand-alone)

T2. Toolkit (API with interfaces)

T3. Source code library (collection of source code files)

These types address different architectural concerns that can be described in different architectural views:

V1. The **Functional view** describes abstractions of system functions provided for the end user. It focuses

on functional requirements and includes functional relationships and decompositions. The view is close to the real-world description of the application domain.

V2. The **Process view** describes processes and threads and their communication as well as the sharing of resources.

V3. The **Development / Code view** describes the system from a developers' view including the structure of the source code. It shows the mapping of functionality to source code.

V4. The **Physical view** describes how the system is deployed in terms of hardware resources.

Based on the characteristics of the three OTS types and the four views, OTS types can be mapped onto views V1 to V4:

$T1 \mapsto V1$. $T2 \mapsto V2$. $T3 \mapsto V3$. $T1 \mapsto V4$.

This mapping might be adjusted for individual projects and objectives and OTS alternatives.

The inclusion of the view concept allows a view-driven multiple OTS selection.

5. The approach

Usually, OTS product information is to be found in different sources (e. g., in-house libraries, market information, vendors, or experts). In the proposed approach, this information is already stored in an OTS product repository. This repository can be considered as an interface or link to OTS information and is not further described in this paper.

The approach has the following properties: 1) It uses atomic requirements as input format. This allows storing OTS product data at the same level of granularity as the actual requirements when doing the selection. 2) It focuses on highly diverse OTS components instead of alternatives with similar functionality. 3) It uses an objective function (see 5.2) to compare OTS components. 4) It creates a set of 4 recommended OTS combination alternatives as output.

5.1. Reuse goals and evaluation criteria

As described in [4], the definition of influence factors, reuse goals and constraints is essential when specifying criteria for OTS evaluation and selection. Therefore, we identify these aspects first in order to define the objective function:

Influencing factors for defining reuse goals and selection criteria are project objectives / constraints, the requirements specification, and the OTS alternatives available.

Reuse goals of using OTS aim to achieve product characteristics (functionality, quality) and to create application architecture artefacts.

Three **Evaluation criteria** based on influencing factors and reuse goals are defined:

a) *Functional criteria* (R_{OTS}): Functional criteria are atomic functional requirements.

b) *Quality criteria* (Q_{OTS}): Non-functional requirements as global concerns (address quality of functionality) are defined as quality criteria.

c) *Architectural criteria* (A_{OTS}): Architectural criteria are architecture attributes and architecture qualities which address properties of the architecture from a technical point of view. Compared to b), these attributes are achieved by the overall architectural structure. Examples are reliability, maintainability, portability, scalability, or performance.

Attributes which are taken into consideration for a specific project may vary.

5.2. The objective function

The objective function O is used to match the problem space with the solution space. Its aim is to maximize the fit and minimize the risk. O requires several definitions described below. The weights in these definitions can be either set by the requirements engineer or assessed by pair-wise comparison and ranking using multi-criteria decision making techniques like the Analytic Hierarchy Process [10]. The definitions are:

a) $R_{ots} = \{r_1, \dots, r_m\}$ with m as the number of functional requirements to be represented in the architecture by OTS. R_{ots} is a subset of the total set of requirements R in the requirements specification. R_{ots} has a weight w to express the importance of requirements: $R_{ots} \cdot w = [0;1]$.

• $\forall r \in R_{ots} \exists$ importance factor $w \in [0;1]$, $\sum_i r_i \cdot w = 1$.

b) $Q_{ots} = \{q_1, \dots, q_n\}$ with n as the number of qualities to be fulfilled in the architecture by OTS. Q_{ots} has a weight w to express the importance of qualities: $Q_{ots} \cdot w = [0;1]$.

• $\forall q \in Q_{ots} \exists$ importance factor $w \in [0;1]$, $\sum_i q_i \cdot w = 1$.

c) $A_{ots} = \{a_1, \dots, a_o\}$ with o as the number of architecture attributes to be fulfilled in the architecture by OTS. A_{ots} has a weight w to express the importance of architecture attributes: $A_{ots} \cdot w = [0; 1]$. $R_{ots} \cdot w + Q_{ots} \cdot w + A_{ots} \cdot w = 1$.

• $\forall a \in A_{ots} \exists$ importance factor $w \in [0;1]$, $\sum_i a_i \cdot w = 1$.

d) $OTS = \{ots_1, \dots, ots_t\}$ as the set of single OTS alternatives with t as the number of OTS alternatives.

• $\forall ots_t \in OTS \exists$ set of requirements $R(ots_t) = \{rs_1, \dots, rs_j\}$ with j as the number of requirements covered by this OTS alternative. Every rs_j has an

assigned risk factor *risk* which is set during OTS assessment.

- $\forall ots_t \in OTS \exists$ set of quality attributes $Q(ots_t) = \{qs_1, \dots, qs_k\}$ with k as the number of quality attributes met by this OTS alternative. Every qs_k has an assigned risk factor *risk* which is set during OTS assessment.
- $\forall ots_t \in OTS \exists$ set of architecture attributes $A(ots_t) = \{as_1, \dots, as_l\}$ with l as the number of architectural attributes listing architecture attributes met by this OTS alternative. Every as_l has an assigned risk factor *risk* which is set during OTS assessment.
- $\forall ots_t \in OTS$ there is an architecture view impacted by this OTS. The view depends on the type of the OTS (see above).

e) The number of all requirements potentially implemented across all OTS components might exceed the number of requirements in R_{ots} . It also can fall short, when not all elements of R_{ots} can be covered by OTS. The same is true for Q_{ots} and A_{ots} and their respective implementations across OTS components:

$ots_1 \cup ots_2 \cup \dots \cup ots_t = R_{ots} \cup diff_rs(OTS)$,
 $ots_1 \cup ots_2 \cup \dots \cup ots_t = Q_{ots} \cup diff_qs(OTS)$,
 $ots_1 \cup ots_2 \cup \dots \cup ots_t = A_{ots} \cup diff_as(OTS)$, with $u = j - m$, $v = k - n$, $w = l - o$ as the difference in requirements, qualities and architecture attributes, respectively. u , v and l can be positive, negative or 0.

f) There is a weight WF for the importance of the fit in the objective function ($WF = [0;1]$).

g) There is a weight WR for the importance of risk in the objective function ($WR = [0;1]$, $WF + WR = 1$).

h) The risk in O is defined based on functionalities, qualities and architecture attributes which are not in R_{ots} , Q_{ob} and A_{ots} but covered by selected OTS. These functionalities, qualities and architecture attributes represent additional risk for which no benefit is achieved. Functionalities, qualities, and architecture attributes which are in R_{ots} , Q_{ob} and A_{ots} are excluded from risk calculation. Their weight already includes the trade-off between anticipated risk and benefit. An example is an OTS implementing remote access features which are not needed by the application under development. This might cause a security risk.

Now, we can define the objective function O :

- (1) $O = O^{max}(R_{ots}, Q_{ots}, A_{ots}, OTS) \times WF \cup O^{min}(R_{ots}, Q_{ots}, A_{ots}, OTS) \times WR$.
- (2) $O^{max}(R_{ots}, Q_{ots}, A_{ots}, OTS) = \text{Max}(\text{fit}) = \max((\sum_{i=1}^m (r_i \cdot w \times r_i \cdot c) \times R_{ots} \cdot w) + (\sum_{j=1}^n (q_j \cdot w \times q_j \cdot c) \times Q_{ots} \cdot w) + (\sum_{l=1}^o (a_l \cdot w \times a_l \cdot c) \times A_{ots} \cdot w))$.
- (3) $O^{min}(R_{ots}, Q_{ots}, A_{ots}, OTS) = \text{Min}(\text{risk}) = \min((\sum_{i=1}^m (rs_i \cdot risk \times rs_i \cdot c) + (\sum_{j=1}^n (qs_j \cdot risk \times qs_j \cdot c) +$

$$(\sum_{l=1}^o (as_l \cdot risk \times as_l \cdot c))).$$

c is a binary variable that determines if a requirement, quality, or architecture attribute is included in the selected combination of OTS components or not. For example, for each tick in Figures 6 – 9, this value is set to 1, otherwise to 0.

The objective function is solved iteratively for all possible combinations of OTS. The 4 best solutions with respect to fit and risk are taken into further consideration. When creating the 4 best solutions, fit and risk constrain each other. Four cases are possible when generating the optimal set of alternatives (Figures 1 – 3):

fit_new > fit_old risk_new < risk_old	
true	false
Case 1 replace(fit_old, fit_new) replace(risk_old, risk_new)	Case 2 keep(fit_old) keep(risk_old)

Figure 1. Case 1 and 2

Case 3 fit_new < fit_old risk_new < risk_old	
true	false
true	false
keep(fit_old) keep(risk_old)	replace(fit_old, fit_new) replace(risk_old, risk_new)
check other cases	

Figure 2. Case 3

Case 4 fit_new > fit_old risk_new > risk_old	
true	false
true	false
replace(fit_old, fit_new) replace(risk_old, risk_new)	keep(fit_old) keep(risk_old)
check other cases	

Figure 3. Case 4

5.3. Steps

Figure 4 shows steps involved in performing the approach. Remaining conflicts between OTS can be resolved by re-implementation, glue ware, etc. [12].

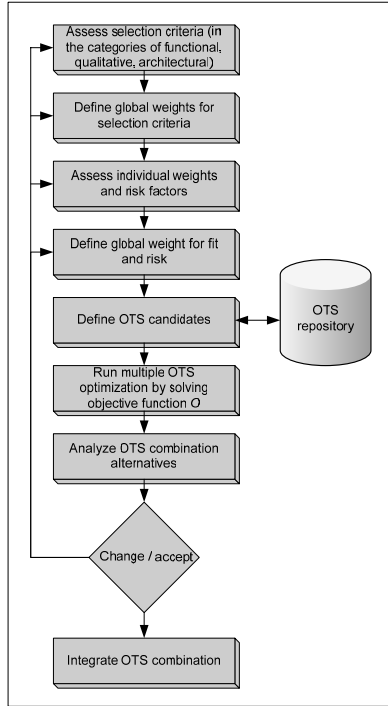


Figure 4. Steps for requirements mapping

6. Example and Evaluation

We implemented the objective function in *MATLAB*. This section illustrates the approach with an example that uses 15 functional requirements, 5 quality attributes and 5 architecture attributes as input for the problem space: $R_{ots} = \{r_1, \dots, r_{15}\}$, $R_{ots.w} = 0.6$, $Q_{ots} = \{q_1, \dots, q_5\}$, $Q_{ots.w} = 0.3$, $A_{ots} = \{a_1, \dots, a_5\}$, $A_{ots.w} = 0.1$. The weights for requirements, qualities, and architecture attributes are defined in Figure 5. *WF* is set to 0.3, and *WR* to 0.7.

A set of possible OTS products is stored in the OTS repository. This repository contains 10 OTS alternatives covering 20 requirements, 10 quality attributes and 10 architecture attributes (Figures 7 - 9). Figure 6 shows the relation between OTS and views. As can be seen from Figures 7, 8, and 9, the OTS are quite diverse. A tick means that the OTS alternative in the row meets the respective requirement, quality, or architecture attribute.

For simplification, we assume a mapping of r_m to rs_m , q_n to qs_n , and a_o to as_o . Shaded columns are not part of the problem space but used for risk calculation.

The initial result for the example is shown in Figure 9, with fit, risk, and views impacted by the selection alternatives. This result is plotted in Figure 11.

Alternative 4 has the highest fit, but also the highest risk. Alternative 1 has the lowest risk, but also the lowest fit. Comparing alternatives 2 and 3 shows that

alternative 3 is preferred due to its higher fit at the same risk. After a trade-off analysis of alternatives either a set of OTS is chosen or the input parameters are changed and another optimization performed. This optimization process ends when several promising sets of OTS are found, which will be analyzed by the developer for the final selection.

R_{ots}	w	Q_{ots}	w	A_{ots}	w
r_1	0.12	q_1	0.40	a_1	0.36
r_2	0.11	q_2	0.32	a_2	0.35
r_3	0.09	q_3	0.18	a_3	0.19
r_4	0.08	q_4	0.06	a_4	0.06
r_5	0.06	q_5	0.04	a_5	0.04
r_6	0.10				
r_7	0.07				
r_8	0.07				
r_9	0.03				
r_{10}	0.05				
r_{11}	0.06				
r_{12}	0.06				
r_{13}	0.02				
r_{14}	0.06				
r_{15}	0.01				

Figure 5. Weights

	Functional View (V1)	Process View (V2)	Development View (V3)	Physical View (V4)
ots_1	✓			
ots_2		✓		
ots_3				✓
ots_4			✓	
ots_5				✓
ots_6	✓			
ots_7		✓		
ots_8		✓		
ots_9			✓	
ots_{10}				✓

Figure 6. Assigning views to OTS

	rs_1	rs_2	rs_3	rs_4	rs_5	rs_6	rs_7	rs_8	rs_9	rs_{10}	rs_{11}	rs_{12}	rs_{13}	rs_{14}	rs_{15}	rs_{16}	rs_{17}	rs_{18}	rs_{19}	rs_{20}
risk	2	3	1	1	2	3	2	1	2	2	3	1	2	1	2	1	2	1	2	1
$R(ots_1)$	✓		✓							✓										
$R(ots_2)$				✓																
$R(ots_3)$					✓															
$R(ots_4)$													✓							
$R(ots_5)$	✓									✓										
$R(ots_6)$				✓			✓													✓
$R(ots_7)$							✓													✓
$R(ots_8)$				✓																
$R(ots_9)$	✓									✓										
$R(ots_{10})$					✓								✓	✓					✓	✓

Figure 7. $R(OTS)$: Assigning requirements to OTS

	qs_1	qs_2	qs_3	qs_4	qs_5	qs_6	qs_7	qs_8	qs_9	qs_{10}	qs_{11}	qs_{12}	qs_{13}	qs_{14}	qs_{15}	qs_{16}	qs_{17}	qs_{18}	qs_{19}	qs_{20}
risk	1	1	1	2	2	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
$Q(ots_1)$	✓	✓					✓													✓
$Q(ots_2)$		✓																		
$Q(ots_3)$			✓																	✓
$Q(ots_4)$																				
$Q(ots_5)$				✓																✓
$Q(ots_6)$					✓															
$Q(ots_7)$						✓														✓
$Q(ots_8)$							✓													✓
$Q(ots_9)$	✓																			
$Q(ots_{10})$		✓	✓				✓	✓	✓	✓										✓

Figure 8. $Q(OTS)$: Assigning qualities to OTS

	as_1	as_2	as_3	as_4	as_5	as_6	as_7	as_8	as_9	as_{10}	as_{11}	as_{12}	as_{13}	as_{14}	as_{15}	as_{16}	as_{17}	as_{18}	as_{19}	as_{20}
risk	2	1	2	1	1	1	1	2	1	1	2	1	1	2	1	1	2	1	2	1
$A(ots_1)$			✓				✓													✓
$A(ots_2)$				✓																✓
$A(ots_3)$	✓																			✓
$A(ots_4)$																				✓
$A(ots_5)$					✓	✓														✓
$A(ots_6)$	✓																			✓
$A(ots_7)$						✓	✓													✓
$A(ots_8)$							✓													✓
$A(ots_9)$								✓												✓
$A(ots_{10})$									✓	✓									✓	✓

Figure 9. $A(OTS)$: Assigning architecture attributes to OTS

When performing the approach we observed the following trends:

- The fewer requirements exist in the problem space, the less OTS alternatives are included in the final combination of OTS.
- The higher the fit, the higher the risk (and vice versa).

- The higher the fit and the higher the risk, the more views are covered (and vice versa).
- Some OTS included in a solution alternative do not impact fit or risk. This happens when either OTS in the repository are similar, or one OTS covers the functionality of two or more other OTS. Such OTS can be removed from a possible solution as they might only increase the complexity of handling more OTS than necessary.

In the example, no view-driven selection was shown. View-driven selection takes views as input and retrieves multiple OTS alternatives which impact these views under the constraint of a high fit and low risk.

	Alternative 1	Alternative 2	Alternative 3	Alternative 4
ots ₁				
ots ₂				
ots ₃				
ots ₄	✓		✓	
ots ₅				✓
ots ₆				
ots ₇				
ots ₈		✓	✓	
ots ₉				✓
ots ₁₀				
Fit	0.033	0.1904	0.2235	0.4276
Risk	2	3	3	6
Views	V3	V2	V2, V3	V3, V4

Figure 10. Solution alternatives

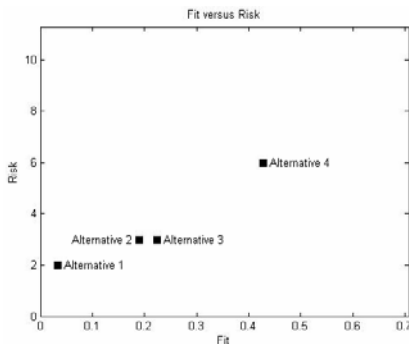


Figure 11. Solution plot

7. Conclusion

The proposed approach allows a partial mapping of software requirements to architectural entities. It helps partially bridging the gap from requirements to architectures. As any knowledge-based approach, the information in the repository must be kept up-to-date. In our case, we must be aware of evolving OTS products. In our future work we will integrate this approach into a larger process framework for the transition from requirements to architectures. We also will design the OTS repository and develop detailed inclusion strategies for adding OTS to the repository.

8. References

- [1] C. Albert and L. Brownsword, "Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview," SEI CMU, Pittsburgh, PA, Technical Report CMU/SEI-2002-TR-009, July 2002.
- [2] L. Bastos, J. Castro, and J. Mylopoulos, "Deriving Architectures from Requirements," in Proceedings of the 14th IEEE International Requirements Engineering Conference, Minneapolis, MA, 2006, pp. 332-333.
- [3] K. Cooper, C. Ramapur, and L. Chung, "A COTS-Aware Requirements Engineering and Architecting Approach," University of Texas, Department of Computer Science, Dallas, TX, Technical Report UTDCS-24-05, December 2005.
- [4] J. Kontio, "OTSO: A Systematic Process for Reusable Software Component Selection," University of Maryland, Institute for Advanced Computer Studies, College Park, MD, Technical Report UMIACS-TR-95-63, December 1995.
- [5] J. Kontio and G. Caldiera, "Defining factors, goals and criteria for reusable component evaluation," in 1996 Conference of the Centre for Advanced Studies on Collaborative research (CASCON 96) Toronto, Ontario, Canada: IBM Press, 1996.
- [6] P. Kruchten, *The rational unified process: An introduction*, 3rd ed. Boston: Addison-Wesley, 2004.
- [7] V. Lakshminarayanan, W. Liu, C. L. Chen, S. Easterbrook, and D. E. Perry, "Software Architects in Practice: Handling Requirements," in Proceedings of the 2006 conference of the Centre for Advanced Studies on Collaborative research (CASCON 2006), Toronto, Canada, 2006, pp. 329-332.
- [8] R. Lichota, "Perspectives on Component-Based Development for Command and Control Systems," in Proceedings of the Workshop on Compositional Software Architectures, Monterey, CA, 1998.
- [9] C. Ncube and N. A. M. Maiden, "PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm," in Proceedings of the 1999 International Workshop on Component Based Software Engineering, Los Angeles, CA, 1999.
- [10] T. L. Saaty, *Decision making for leaders: The analytic hierarchy process for decisions in a complex world*. Pittsburgh, PA: RWS Publications, 2001.
- [11] M. R. Vigder, T. McClean, and F. Bordeleau, "Evaluating COTS Based Architectures," in LNCS 2580 - COTS-Based Software Systems: Second International Conference (ICCBSS), H. Erdogmus and T. Weng, Eds. London, UK: Springer-Verlag, 2003, pp. 240-250.
- [12] D. Yakimovich, G. H. Travassos, and V. Basili, "A Classification of Software Components Incompatibilities for COTS Integration," in Proceedings of the Twenty-Fourth Annual Software Engineering Workshop, Greenbelt, MD, 1999.