Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 16 (2013) 216 – 225

Procedia Computer Science

Conference on Systems Engineering Research (CSER'13)
Eds.: C.J.J. Paredis, C. Bishop, D. Bodner, Georgia Institute of Technology, Atlanta, GA, March 19-22, 2013.

Model-Based Systems Engineering Design and Trade-Off Analysis with RDF Graphs

Nefretiti Nassar and Mark Austin*

Graduate Student, Institute for Systems Research, University of Maryland, College Park, MD 20742.
Associate Professor, Department of Civil and Environmental Engineering, and Institute for Sytsems Research, University of Maryland, College Park, MD 20742.

Abstract

Within the Semantic Web community, resource description frameworks (RDF) and logical reasoning engines work together to provide semantic support and reasoning for Web applications. This paper explores the use of RDF graphs for the representation of graphs of requirements and specification of design component properties. We show that the component selection design problem and associated trade-space analysis can be cast as a sequence inference analyses on RDF graphs. Inference procedures are provided for assessment of requirements in terms of component attribute values, identification of compatible component interface pairs, component selection to meet the system architecture requirements, and computation of system cost, performance and reliability. Our prototype implementation makes extensive use of Java and Python, and focuses on the satisfaction of requirements and component selection for a home theater design problem.

Selection and/or peer-review under responsibility of Georgia Institute of Technology

Keywords: semantic web, graph-based design; inference rules; system design; trade-off analysis.

1. Introduction

The Semantic Web is important to the Systems Engineering community because it provides formalisms (i.e., models and tools) for sharing and reasoning with data on the Web. As companies move toward the team-based development of projects and products, having Web access to design specifications and component specifications adds value to business operations. We assert that Semantic Web concepts and technologies can also provide considerable assistance in the model-based systems engineering and design of modern-day systems, which are undergoing a series of radical transformations to meet performance, quality, and cost constraints. To keep the complexity of technical concerns in check, system-level design methodologies are striving to separate and simplify concerns (i.e., to allow more efficient exploration of the space of potential design alternatives), improve economics through reuse at all levels of abstraction, and employ formal design representations that enable early detection of errors and multi-disciplinary design rule checking. We believe that methodologies for strategic approaches to design

1877-0509
Selection and/or peer-review under responsibility of Georgia Institute of Technology
doi:10.1016/j.procs.2013.01.023

will employ semantic descriptions of application domains, and use ontologies and rule-based reasoning to enable validation of requirements, automated synthesis of potentially good design solutions, and communication (or mappings) among multiple disciplines [1-3]. Semantic Web-based technologies can play a central role in the design of tools to support these design methodologies. Present-day systems engineering methodologies and tools, including those associated with SysML, are not designed to handle projects in this way.

1.1. Scope and Objectives

The standard suite of languages, models, and tools for the implementation of Semantic Web applications includes XML (eXtended Markup Language), RDF (Resource Description Framework), OWL (Web Ontology Language), Jess, Jena, Protégé and SWRL (Semantic Web Rule Language), among others [4-9,11]. Because each aspect of Semantic Web technology is designed to serve a specific purpose, working applications nearly always correspond to an integration of languages, models and tools. Our hands-on experience in developing applications that involve Semantic Web technologies tells us that when some of the models and tools are prototypes or tentative proposals (e.g., SWRL and Protégé), the challenge in getting things to work properly is nearly always significant, and perhaps a lot more difficult than really necessary. Therefore, in an effort to streamline and simplify computational design support, in this paper we explore the use of RDF graphs for the representation of requirements and design component properties, and Python for the implementation and sequencing of logical reasoning and inference mechanisms. This approach is inspired, in part, by reference [10]. From a model-based systems engineering perspective, the use of RDF graphs makes sense because they are general in their ability to represent data in environments that will be subject to change. Python is a mature and well-known scripting language that provides for easy implementation of inference mechanisms. Two research questions of interest are: (1) What are we giving up by removing OWL, Jess, Jena, Protégé and SWRL from the formulation of problem solutions, and (2) Does it matter for model-based systems engineering?

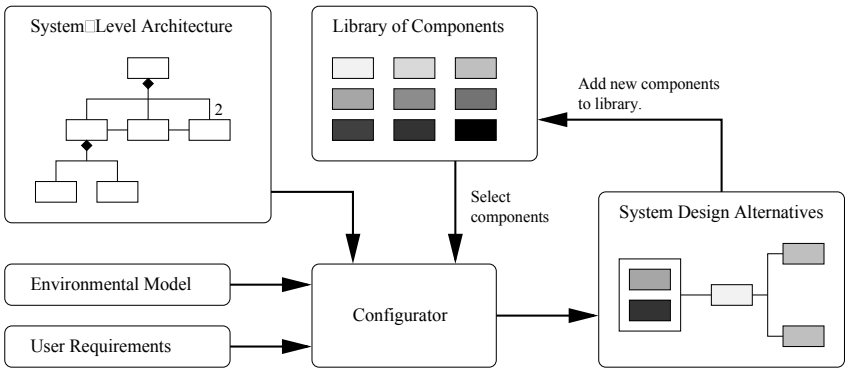

Fig. 1. Schematic of the component-selection design problem

To explore these issues, our prototype implementation focuses on the satisfaction of requirements and components selection for a home theater design problem. As illustrated in Fig. 1 we will assume that the system-architecture is fixed and that components can be selected from a component library. We show that the component selection design problem and ensuring trade-space analysis can be cast as a sequence inference analyses on RDF graphs. Inference procedures are provided for assessment of requirements in terms of component attribute values, identification of compatible component interface pairs, component selection to meet the system architecture requirements, and computation of system cost and performance. An algorithm for identification of Pareto-Optimal design solutions has been developed [3].

2. Background Review

2.1. Graph-Based Modeling and Systems Design

Graph structures are essential for the representation of system requirements and system architectures. When requirements are organized into levels for team development, graph structures are needed to describe comply and define relationships among requirements (terminology such as incoming and outgoing requirements is sometime used). A parent requirement may have multiple derived children requirements, and a derived child requirement may have multiple parents. Individual requirements are linked together using graph structures. Depending on the requirements context, they need to support the allocation of requirements onto a number of other system modeling entities like parts (components), functions and interfaces.

2.2. Semantic Web Vision and Technical Infrastructure

In his original vision for the World Wide Web, Tim Berners-Lee described two key objectives [6]: (1) To make the Web a collaborative medium, and (2) To make the Web understandable and, thus, executable by machines. During the past twenty years the first part of this vision has come to pass -- today's Web provides a medium for presentation of data/content to humans. Machines are used primarily to retrieve and render information. The Semantic Web aims to produce a semantic data structure that allows machines to access and share information, thus constituting a communication of knowledge between machines, and automated discovery of new knowledge. Realization of this goal will require mechanisms (i.e., markup languages) that will enable the introduction, coordination, and sharing of the formal semantics of data, as well as an ability to reason and draw conclusions (i.e., inference) from semantic data obtained by following hyperlinks to definitions of problem domains (i.e., so-called ontologies).

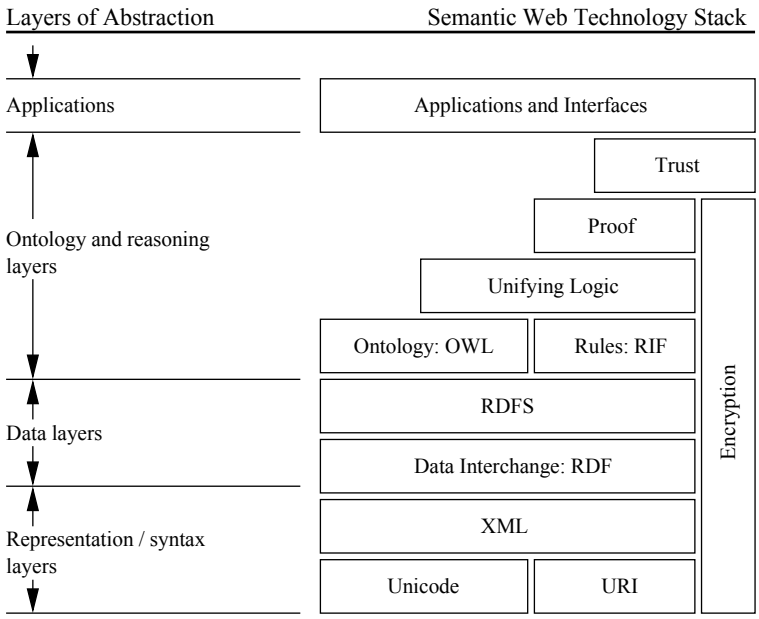

Fig. 2. Layers of abstraction and technology in the Semantic Web

Fig. 2 illustrates the technical infrastructure that supports the Semantic Web vision. Each new layer builds on the layers of technology below it. The bottom layer is constructed of Universal Resource Identifiers (URI) and Unicode. URIs are a generalized mechanism for specifying a unique address for an item on the web. The eXtensible Markup Language (XML) provides the fundamental layer for representation and management of data that can be organized into hierarchical relationships. However, a common engineering task is the synthesis information from multiple data

sources. This can be a problematic situation for XML as a synthesized object may or may not fit into a hierarchical (tree) model. A graph, however, can, and thus we introduce the Resource Description Framework (RDF). RDF is a graph-base data model for describing the relationships between objects and classes (i.e., data and metadata) in a general but simple way, and for designating at least one understanding of a schema that is sharable and understandable. The graph-based nature of RDF means that it can resolve circular references, an inherent problem of the hierarchical structure of XML.

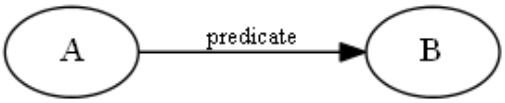

Fig. 3. Example of an RDF triple where A is the subject, the predicate is a verb, and B is an object.

In practical terms (see Fig. 3), English statements are transformed into RDF triples consisting of a subject (this is the entity the statement is about), a predicate (this is the named attribute, or property, of the subject) and an object (the value of the named attribute). Subjects are denoted by a URI. Each property will have a specific meaning and may define its permitted values, the types of resources it can describe, and its relationship with other properties. Objects are denoted by a ``string'' or URI. The latter can be web resources such as requirements documents, other Web pages or, more generally, any resource that can be referenced using a URI (e.g., an application program or service program). A set of related statements constitute an RDF graph. The ontology, logic, proof and trust layers introduce vocabularies, logical reasoning, establishment of consistency and correctness, and evidence of trustworthiness into the Semantic Web framework. Class relationships and statements about a problem domain can be expressed in the Web Ontology Language (OWL) [7]. Recently, rule languages such as SWRL (Semantic Web Rule Language) have been developed to provide designers with mechanisms to reason with data and class relationships [11].

3. Design Methodology

In our work, RDF graphs are used to model individual design requirements, graphs of requirements, the characteristics of individual components, and graphs of design components. Synthesized graph models are employed for libraries of design requirements and components, as well as the mapping of requirements to design solutions. We use Python to script and sequence the inference rules, computation of Pareto-Optimal design solutions, and generation of two-dimensional plots for the trade-space (e.g., cost versus performance).

3.1. Component-Selection Problem

The component-selection problem can be stated as follows: We wish to choose a subset of components from a library of components to satisfy the requirements of a system-level architecture, and component- and system-level requirements. As illustrated in Fig 1., we will assume that a representation for component specification exists, and that a component library has been designed as built. Generally speaking, two outcomes to the component selection problem are possible. The first possibility is that a search procedure will find one or more combinations of components that satisfy all of the architectural, functionality and performance requirements. In such cases, we will choose a subset of feasible designs that maximize performance, minimize cost, and so forth. The second class of outcomes occurs when the design requirements are stated in such a way that no feasible designs exist. This problem can be solved by either relaxing the requirements (i.e., values of the inequality constraints representing the requirements), or by developing new components that will have superior performance and/or extended functionality. Our research objective is to devise strategies for component selection during the early stages of design, where the

number of components is unlikely to be excessively large.

3.2. Synthesis of Design Solutions from RDF Graph Representations of Requirements and Design Components

Fig. 4 is a flowchart of activities for problem definition with RDF graphs followed by a series of inference-rule driven graph transformations designed to generate feasible design solutions.

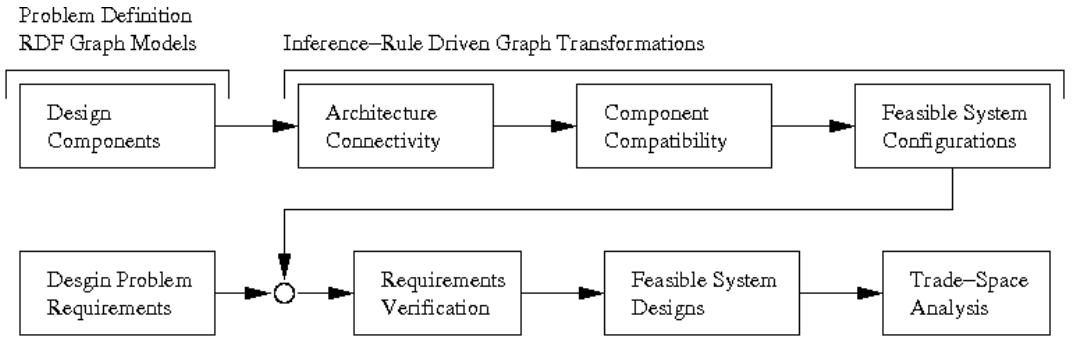

Fig. 4. Flowchart of activities for problem definition with RDF graph models followed by inference-rule driven graph transformations.

The processing pipeline begins with RDF graph representations of individual requirements and design components. Then, customized inference rules are created in Python for: (1) the representation of the system architecture, (2) identification of pairwise component compatibilities, (3) the identification of feasible system configurations, (4) validation of design requirements, (5) identification of feasible system designs, and (6) trade-space analysis. It is important to note that steps (1), (2) and (3) only relate to the system architecture definition, and properties and compatibility of the design components. Hence, at the front-end of the design synthesis, there is no need to include the design problem requirements. The feasible system configurations computed at step (3) are configurations of components that satisfy the architecture descriptions – they may not also satisfy the design problem constraints. The feasible system designs computed at step (5) are feasible configurations that satisfy all of the design requirements. Finally, trade-space analysis is conducted to identify sets of Pareto-Optimal design solutions – the details of a simple algorithm and Python code to compute the Pareto-Optimal design solutions can be found in Nassar [3].

4. Home Theater Design Problem

The home theater design problem was posed by David Everett at NASA Goddard Space Flight Center as an exercise in understanding how requirements should be written and organized to support the purchase of a home theater system. The long-term hope is that systems engineers will be able to search for components on the Web that satisfy a specific set of technical specifications.

4.1. Problem Description

Suppose that we wish to purchase a home theater system that satisfies a budgetary constraint (total cost < $2,100) and as illustrated in Figs. 5 and 6 below, has an amplifier, speakers and a flat screen TV. Development of the design requirements begins with a statement of need (and initial requirements reflecting the statement of need) and evolves into three levels of requirements. The Level 1 requirements are the initial requirements (e.g., R01: I need a home theater system). The Level 2 requirements serve the purpose of defining a detailed agreement between the customer and supplier (e.g., R03: The theater system shall have a large display screen). The Level 3 represents are the

component-level requirements and are cast as inequality constraints written in terms of the component attribute values (e.g., R08: The width of the screen shall be at least 3 ft, R09: The height of the screen shall be at least 2ft). In other words, they contain quantitative elements that can constrain the selection of components from a database. The problem formation addressed here is simplified in the sense that only the Level 3 requirements are evaluated quantitatively. Evaluation of the Level 1 and Level 2 requirements occurs through the logical satisfaction of the sets of lower-level requirements.

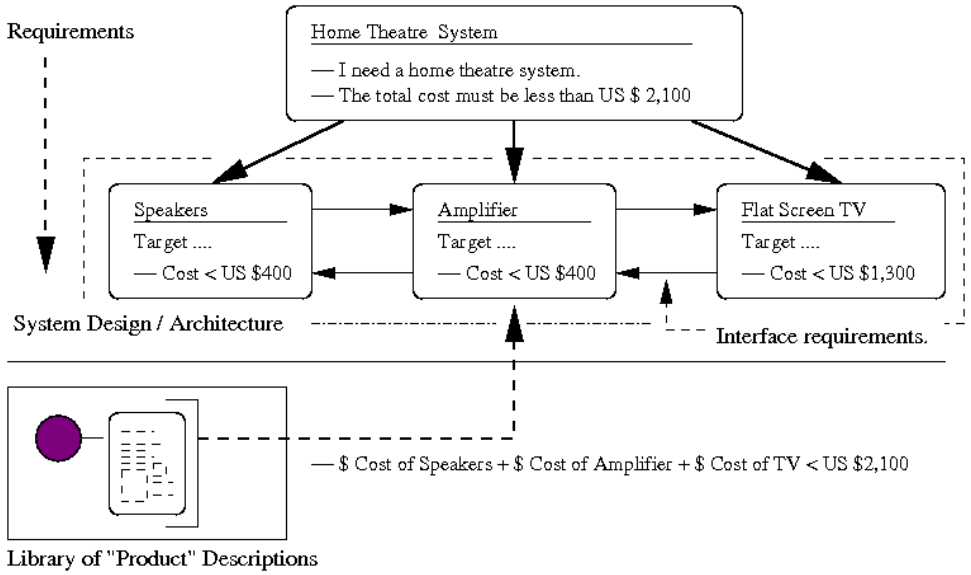

Fig. 5. Flow down of requirements to system architecture definition. Assembly of the home theater system from library components.

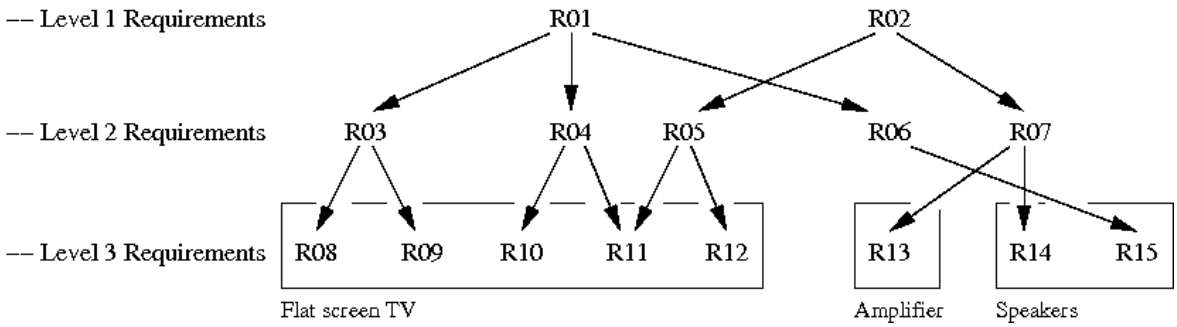

Fig. 6. Definition and organization of requirements into three levels.

The design component library [12,13] contains three television components (Sony, Samsung, and Sony), three amplifier components (Bose, Polk, and Klipsch) and three speaker components (Polk, Klipsch, and Bose). As illustrated in Fig. 5, amplifier components will connect to speaker and TV components. The maximum number of potentially good design configurations is 27.

4.2. Modeling Requirements and Design Components as RDF Graphs.

The requirements and design components models were initially developed as objects in Java, exported to Excel in a comma-separated value (CSV) format, and then read into a simple graph triple-store for RDF statements, implemented in Python.

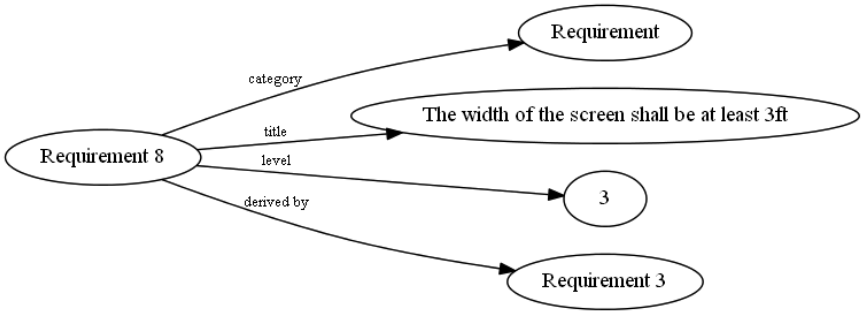

Fig. 7. RDF graph model for Requirement 8

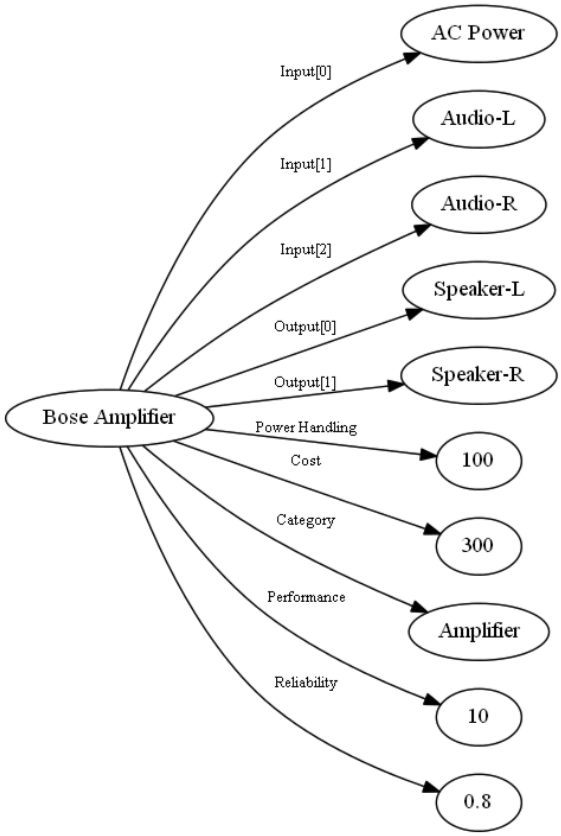

Fig. 8. RDF graph model for a Bose amplifier component.

For systems engineering applications, RDF graph representations provide a desirable balance of and expressiveness and flexibility. Fig 7 shows, for example, a collection of RDF triples – subject, predicate, object – organized into a graph representation for Requirement 8. In each case, the subject is Requirement 8. The predicates are: category, title, level and derived by. String representations for the triple objects are shown along the right-hand side of Fig. 7.

Fig. 8 shows an RDF graph representation for a Bose Amplifier component. The RDF triples systematically describe the input/output capabilities of the amplifier, its power handling capabilities and cost. Flexibility stems from the use of graphs – no need to conform to a table data structure – and the use of Python for the implementation of the simple graph class.. For our purposes, inference rules a useful because they can deduce whether or not inequality constraints have been satisfied, and to infer relationships between design components and requirements. The design component and requirements libraries (see the left-hand side of Fig. 4) were created by simply loading CSV representations for all requirements and design components into merged graphs for the requirements and design components, respectively. Each design requirement is translated into a Python inference rule according to the description, level, and dependencies among the requirements (see Fig. 6).

4.3. Synthesis of Feasible System Configurations

The system architecture rules declare that television components can only connect to amplifiers. Speakers can only connect to amplifiers. For example the script of Python implements a television-to-amplifier rule:

```
class televisionToAmplifier(InferenceRule):

      def getqueries(self)
          tvToAmp=[ ('?x','category','television'), ('?y','category','amplifier')]
          return [tvToAmp]

      def _maketriples(self,x,y):
          return [ ( x, 'connects to', y ) ]
```

The rule checks to see if ``x'' and ``y'' are televisions and amplifiers, then if true, creates a new relationship ``(television x) connects to (amplifier y).'' The component compatibility rules establish whether or not different component types are compatible based upon the outputs of the first matched against the inputs of the second. For the synthesis of home theater design solutions, amplifier-to-television and amplifier-to-speaker compatibility is prerequisite to a feasible system configuration. With 3 amplifiers, 3 speakers and 3 televisions in the components library, there are potentially 27 configurations that could work – however, limitations on the input/output relationships reduce the number of potentially good designs from 27 to 18.

4.4. Quantitative Evaluation of Design Requirements

The quantitative evaluation of design requirements involves development of inference rules for the evaluation of level 3, level 2 and level 1 requirements, i.e.,

```
requirement15 = Req15() mergegraph.applyinference(requirement15)

requirement14 = Req14() mergegraph.applyinference(requirement14)

requirement13 = Req13() mergegraph.applyinference(requirement13) … etc.
```

Inference rules for the level 3 requirements are derived directly from the inequality constraints defined in the requirements statement. For example, the inference rule for R15 evaluates speaker design options to see if their ``power handling'' capabilities meet requirements R15. When an inequality constraint is satisfied, a new ``x satisfies y'' triple is added to the merged graph of requirements and design constraints. After applying the level 3 requirement inference rules to the merged graph, the graph has 100 unique vertices and 308 edges. The size of the graph in terms of the number of vertices has increased by the addition of a satisfied vertex. However, the order of the graph in terms of the number of edges has increased by a factor of 30 new relationships. After applying the level 2 requirement inference rules to the merged graph, the graph has 100 unique vertices and 319 edges. The size of the graph in terms of the number of vertices remained the same. But, the order of the graph in terms of the number of edges has increased by a factor of 11 new relationships. After applying the level 1 requirement inference rules to the merged graph, the graph has 100 unique vertices and 321 edges.

4.5. Synthesis of System-Level Design Alternatives

The system design rule establishes relationships between the design components and the design requirements. As illustrated in the abbreviated script of output, there are nine system level designs that satisfy all of the requirements:

```
System Design 1 :  [u'Lg Television', u'Bose Amplifier', u'Polk Speaker']
System Design Cost = $ 2000, Performance = 23, Reliability = 0.448.

… output removed ….

System Design 9 :  [u'Sony Television', u'Polk Amplifier', u'Bose Speaker']
System Design Cost = $ 1878, Performance = 28, Reliability = 0.729.
```

The highest performing system-level design has a value of 30 with an associated cost of USD $1,828 and a reliability of 0.648. This system-level design includes a Sony television, Bose amplifier, and a Bose speaker. The most reliable system-level design has a value of 0.729 with an associated cost of USD $1,878 and a performance of 28. This system-level design includes a Sony television, Polk amplifier, and a Bose speaker. The least expensive system-level design cost USD $1,828 with an associated performance of 30 and a reliability of 0.648. This system-level design includes a Sony television, Bose amplifier, and a Bose speaker. The design with the highest performance level is also the least expensive. Thus, the tentative conclusion is as follows: the best system-level design will include a Sony television, Bose amplifier, and a Bose speaker.

4.6. Trade-Space Analysis

Trade-space analysis is based upon a systematic comparison of the feasible system designs measured with respect to cost, performance and reliability. We have developed a simple algorithm to automatically identify the Pareto-Optimal design solutions within a space of two design criteria (e.g., cost versus reliability as shown in Fig. 9).

5. Conclusions and Future Work

In this paper we have presented work that investigates the use of RDF graphs and inference rule transformations implemented in Python for the automated synthesis and evaluation of design solutions. In addition to validating

whether or not such an approach would work, we wanted to start to understand the pros/cons of working with RDF in lieu of OWL and all of its associated tools and models. Our proposed approach is simple. For example, in contrast to Jena and OWL, understanding exactly what is stored in the RDF graphs is straightforward. The RDF graphs are an order of magnitude smaller than their OWL counterparts. On the other hand, coding inference rules to evaluate requirements is less than ideal. Maybe this problem can be overcome by redesigning the RDF graph models so that they are a little more general, but allow for fewer more powerful inference rules?

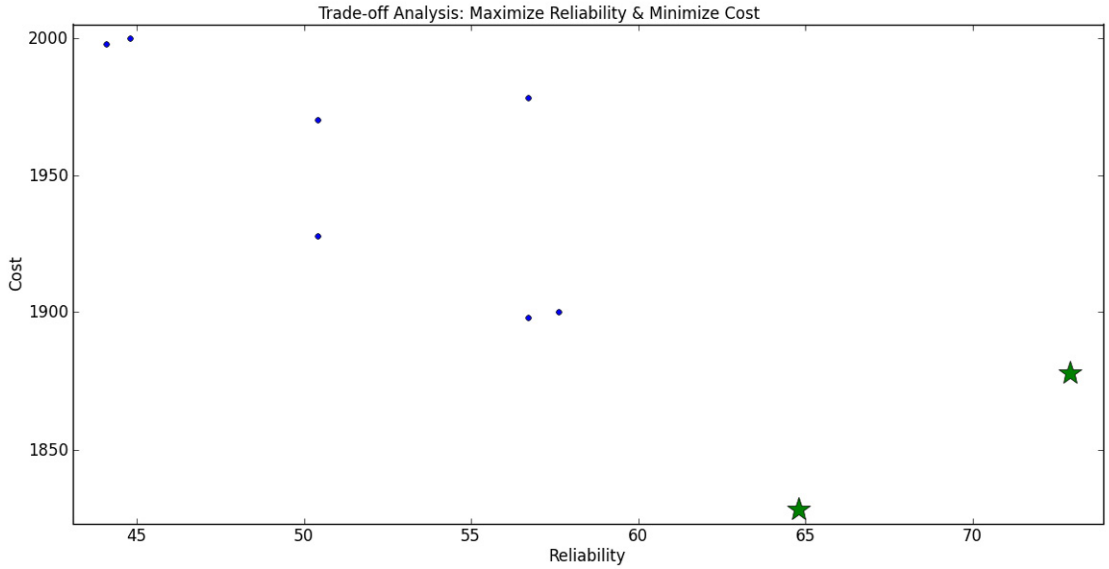

Fig. 9. Trade-off analysis for cost versus reliability.

References

1. M.A. Austin, V. Mayank, and N. Shmunis, Ontology-Based Validation of Connectivity Relationships in a Home Theater System, International Journal of Intelligent Systems, 21(10) 1111-1125, October, 2006.
2. M.A. Austin, V. Mayank, and N. Shmunis, PaladinRM: Graph-Based Visualization of Requirements Organized for Team-Based Design, Systems Engineering, 9(2): 129-145, May 2006.
3. N. Nassar, Systems Engineering Design and Tradeoff Analysis with RDF Graph Models, M.S. Thesis, Insitute for Systems Research, University of Maryland, College Park, MD 20742, December 2012.
4. RDF: Resource Description Framework. See http://www.w3.org/RDF/ (Accessed December 2012).
5. G Klyne and J.J. Carroll, Resource Description Framework (RDF): Concepts and Syntax. See http://www.w3.org/TR/rdf-concepts/ (Accessed December 2012).
6. T Berners-Lee, J. Hendler, and O. Lassa, The Semantic Web, Scientific American, 35-43, May 2001.
7. Web Ontology Language (OWL). See http://www.w3.org/TR/owl-ref/, (Accessed September 2012).
8. Protege Ontology Editor and Knowledge Acquisition System, See http://protege.stanford.edu
9. Jess – The Expert System Shell for the Java Platform. See http://herzberg.ca.sandia.gov/jess/, 2003.
10. T. Segaran, J. Taylor, and C. Evans, Programming the Semantic Web, O'Reilly, Beijing, 2009.
11. Horrocks, et al., SWRL: A Semantic Web Rule Language combining OWL and RuleML, W3C Member Submission, 2004.
12. Best Buy – Computers, Video Games, TVs, Cameras, Appliances. See: http://www.bestbuy.com, 2012.
13. Polk Audio. See http://www.polkaudio.com, 2012.