Using ontology to support development of software architectures

A. Akerman
J. Tyree

In this paper we propose an approach to software development that focuses on architecture decisions and involves the use of ontology. In this approach the architecture is captured by an instance of an ontology. The ontology has four major components: architecture assets, architecture decisions, stakeholder concerns, and an architecture roadmap. We illustrate our approach through a case study involving a real-time credit-approval system and the use of Protégé, an open-source ontology development tool.

INTRODUCTION

The *IEEE Recommended Practices for Architectural Description of Software-Intensive Systems* (IEEE-1471)[1] provides a conceptual framework for documenting software architectures, in which the documentation is organized as a set of architecture views. Each view addresses one or more of the concerns of stakeholders, the parties involved in the development project, such as executives, software developers, and software architects. Traditional view-based approaches, such as the Reference Model for Open Distributed Processing (RM-ODP),[2] the 4+1 View Model,[3] and IBM's Rational* Unified Process*[4] generally adhere to this standard. For example, RM-ODP organizes the architectural description into Enterprise, Information, Computational, Engineering, and Technology views.

In contrast, in *Architecture Decisions: Demystifying Architecture*,[5] we argue that architecture decisions are the primary representation of architecture. Rather than starting with view construction, we first document explicit architecture decisions and then complement those with views. In this paper we extend our previous work and submit that software architecture should foundationally be captured and maintained as an instance of an architectural ontology. A similar approach has been used at the University of California at Irvine.[6]

By definition, *ontology* is an explicit formal specification of the *concepts* (also referred to as *classes*) in a domain and the relations among them.[7] Classes contain *properties* (also known as *slots*), which describe various features and attributes of the class.[8] Ontologies are used in various application domains

to facilitate a common understanding of the information structures in a domain and to enable reuse of domain knowledge. We view ontology as a powerful mechanism that can play a similar role in the world of software architecture.

An architectural ontology provides a common vocabulary that enables the level of precision needed for making effective architecture decisions. Our proposed ontology is composed of four segments: (1) architecture assets, such as subsystems, components and interfaces, (2) architecture decisions, (3) stakeholder concerns, and (4) an architecture roadmap that describes *which* aspects of the architecture should be developed and *when* they should be tackled. The work we present here offers the following benefits beyond our original approach:

- *The approach ensures the architect is focused on what is important*. Just as viewpoints, or templates for views, provide guidelines for architectural descriptions, architecture-related decision making needs appropriate guidelines to focus attention on those assets that are important to the organization.
- *The ontology-based approach provides a common vocabulary to enhance precision and clarity*. One benefit of architecture-modeling approaches is the added degree of precision they provide. Similarly, an architectural vocabulary provides a higher level of precision for decision making.
- *Tools that provide repository support are available*. In large projects we make hundreds of decisions, which need documenting. Also required is support for navigation and reuse. From the various tools available we have chosen Protégé, an open-source ontology development tool.
- *The approach supports impact analysis*. Architecture decisions change due to changes in business needs, product experience in the field, changes in schedules, and so on. The approach and the supporting tools enable us to perform what-if analyses, which lead to improved architecture decisions.
- *The approach supports on-demand view creation*. It has proven difficult to predetermine which views to construct before the architecture is documented. This approach gives us the ability to create views on demand from a structured repository.
- *The approach supports the temporal mapping of the development of the architecture*. Stakeholders often require a time-based view of the evolution of the architecture over a number of releases, which we provide through an architecture roadmap.

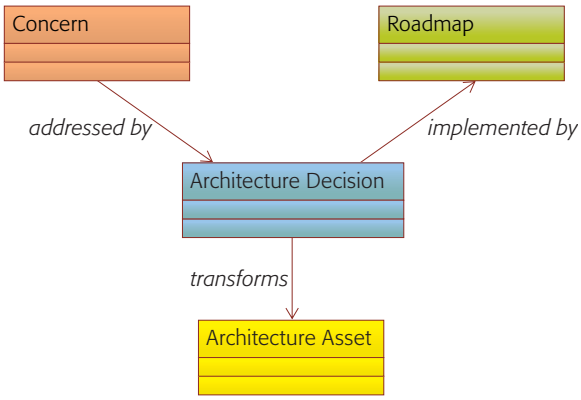

Figure 1
High-level view of ontology

The rest of this paper is organized as follows. In the next section we describe our ontology-based approach to architecture development. In the section that follows we illustrate our approach through a case study that involves a credit-approval system and the use of Protégé, an open-source ontology development tool. We then discuss the results of our case study and the lessons learned, after which we describe related research and position our work in its context. We conclude with a summary and ideas for future work.

OUR APPROACH

Our approach is built on an ontology that connects stakeholder concerns, architecture decisions, architecture assets, and an implementation roadmap. The concerns of the various stakeholders (strategic business needs, business risks, specific functional requirements, etc.) drive architecture entities and are captured as a set of key decisions. The decisions can be viewed as making changes to the various assets in the information technology (IT) environment, such as systems, interfaces, nodes, and components. For example, a decision may call for decommissioning of an existing system or for the development of a new interface. These changes are carried out based on a technology transformation roadmap.

A high-level view of this ontology is shown in ***Figure 1***; in this paper, we use the UML** (Unified Modeling Language**) notation in the diagrams

representing ontology structures. The class Concern has the slot `addressed by`, whose value is class Architecture Decision; class Architecture Decision in turn has slots `transforms` and `implemented by`, whose values are classes Architecture Asset and Roadmap.

We develop this ontology in five steps: (1) capturing stakeholder concerns, (2) analyzing the current architecture, (3) defining the target architecture, (4) conducting a gap analysis and producing the roadmap, and (5) validating the architecture. These are the same steps we described in *An Architecture Process for System Evolution*,[9] and the process is similar to the one used by other methodologies such as The Open Group Architecture Framework (TOGAF**).[10]

In the next section we illustrate our approach through a case study in which we capture the ontology using Protégé, an open-source ontology development tool from Stanford University.[11] Protégé provides a graphical user interface for modeling classes, properties, and relations. Protégé generates interactive forms for domain experts to enter ontology data, which is then validated by the tool. The tool comes with a large collection of plug-ins that query and visualize ontology data. Models (classes and instances) are loaded and saved in various formats, including XML (Extensible Markup Language) and RDF (Resource Description Framework). Protégé models may be stored in any database supporting JDBC** (Java Database Connectivity).

CASE STUDY

In *Architecture Decisions: Demystifying Architecture*,[5] a case study introduced a complex real-time credit-approval process in a predominately batch environment. We make use of the same scenario here and present a case study in which our ontology-based approach to architectural development is illustrated. In the section "Discussion," we describe the way in which our previous results have been further improved.

The management of a fictional large financial organization (the company) has determined that it has to provide an immediate response to customers when they apply for a new financial product (credit card, loan, savings account, etc.). The current approval process is complex; it involves batch processing of data; and the customers are notified by mail. A system developed in house (system A) performs the batch processing. The processing flow and the business rules are "hard coded," and there are few configuration options. The company has recently deployed a commercial off-the-shelf (COTS) system (system B) to handle the approval process for a certain type of financial product. The COTS package contains a flexible workflow engine that allows non-IT personnel to enter or modify business rules. This solution works in both batch mode and interactive mode, but extending support for additional product types requires significant customization work. Both systems interact with numerous client systems. None of these interactions happens in real time, as processing involves file transfer via FTP (File Transfer Protocol). Although the company has an internally developed middleware platform that is used by many customer-facing applications, neither system A nor system B is configured to use this middleware.

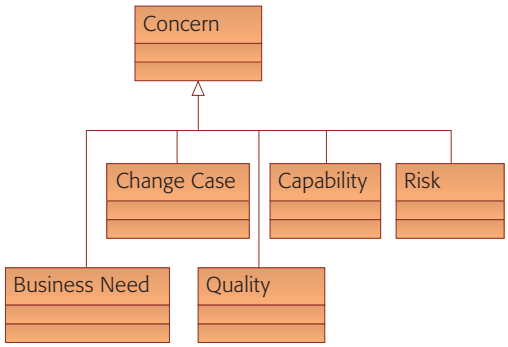

Figure 2
Concern segment of ontology

Capturing stakeholder concerns

We articulate the business vision as a set of stakeholder concerns, which are represented by class Concern. ***Figure 2*** illustrates the Concern branch of the ontology, which consists of class Concern and its five subclasses: Business Need (business goals, objectives, or issues), Risk, Change Case (future requirements), Quality (nonfunctional requirements such as performance, reliability, etc.), and Capability (functional features of the system). The arrow pointing to class Concern represents the relation `subclass of`. We capture all the "architecturally significant" concerns, both pre-existing and new, in Protégé. ***Figure 3*** shows how the concerns appear on the screen. Here are some instances of Concern that we have identified in our case study.

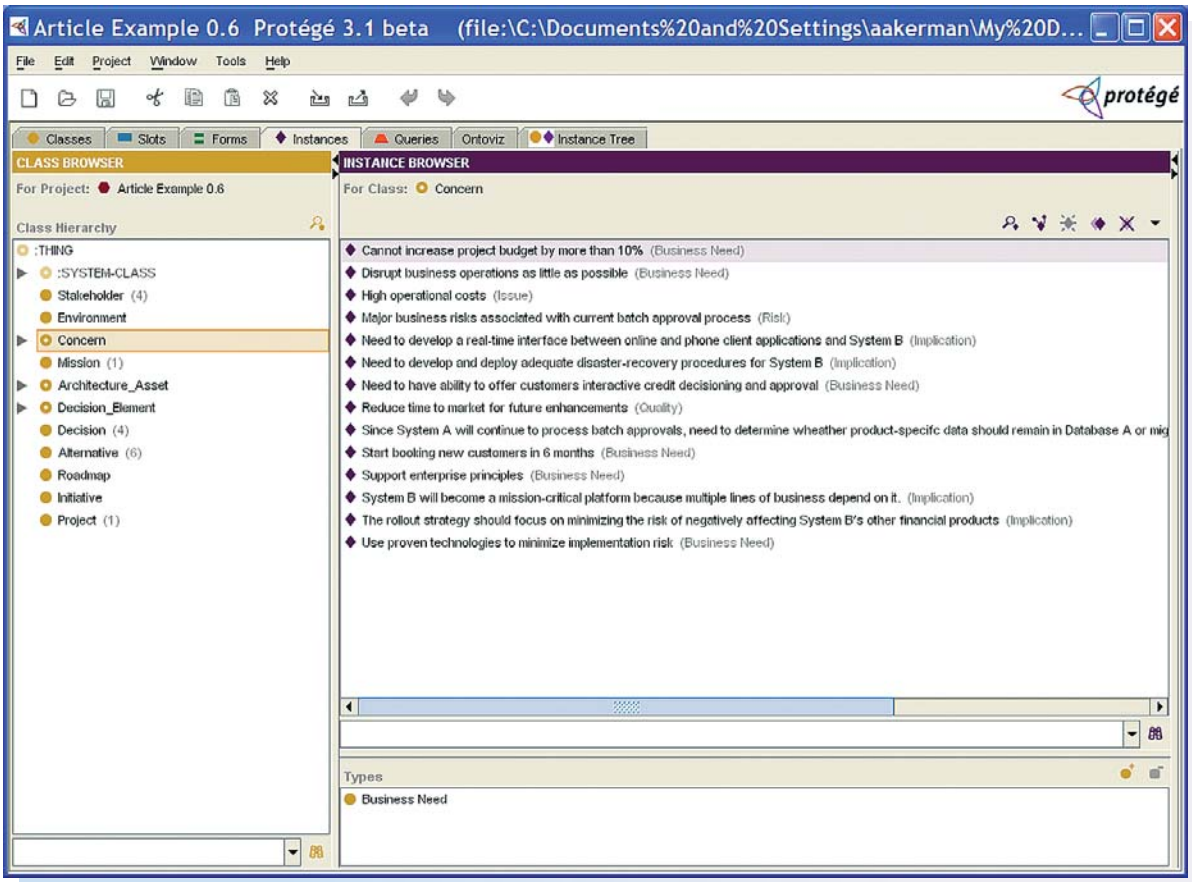

Figure 3
Class Concern as viewed in Protégé

- Offer customers a real-time credit-approval service (Business Need).
- Make credit-approval service available within six months (Business Need).
- Limit budget overrun to 10 percent (Business Need).
- Keep disruption of business operations to a minimum (Business Need).
- Ensure that time to market for future enhancements is under two weeks (Quality).
- Offer credit-approval service to external partners (Change Case).

Analyzing the current architecture

We almost never develop architecture from scratch. The existing environment provides a baseline (which includes architecture assets such as systems, components, nodes, and interfaces) on which the new system is built. We begin the process by populating our ontology with the instances of class Architecture Asset that correspond to the current environment. We make use of a simplified version of the IBM Architecture Description Standard[12,13] for this segment of the ontology (***Figure 4***). All classes depicted in Figure 4 are subclasses of Architecture Asset; the inheritance relation is omitted for clarity.

We identify five instances of class System: Credit Approval System, Call Center Desktop System, Internet Customer Servicing System, Enterprise Integration System, and Analytical Environment. In addition, we define 10 instances of class Subsystem, 15 instances of class Component and nine instances of class Interface. Although the actual environment contains many more architecture assets, we limit our scope to the ones that participate in the credit approval process. ***Figure 5*** shows a traditional components-and-connectors diagram for this portion of the current environment as an example of an

on demand view, which could be used to educate new team members. This view is generated using Ontoviz, a Protégé plug-in.

As we populate the ontology with architecture assets, we come across existing concerns, and we capture these as instances of Business Need and Risk. In our case study we identified the following additional concerns:

- Major business risks associated with lack of a real-time credit-approval offering (Risk)
- High operational costs of current system (Business Need)

Defining the target architecture

Architecture results from architecture decisions. Kruchten[14] lays the framework for an ontology of software-architecture design decisions. We heavily leverage his work by making use of its key concepts and connecting them to the other segments of our ontology. ***Figure 6*** illustrates the Architecture Decision segment of the ontology. We make an architecture decision by selecting an appropriate alternative (represented as an instance of class Alternative and the relation `has`), which in turn has implications (represented as instances of class Implication and relation `implies`). Figure 6 also shows the class Concern-Alternative Relationship, which establishes a linkage through a selected alternative between an architecture decision and a stakeholder concern. Following a soft-goal framework,[15] property `addresses as soft concern` has values `makes`, `helps`, `neutral`, `hurts`, and `breaks`. This class embodies the extent to which an alternative addresses a concern. Not all concerns affect architecture decisions. In practice, we only catalog the architecturally significant concerns, which means that every concern maps to at least one alternative, hence, at least one architecture decision. For each concern, an assessment is made across selected alternatives as to how adequately the concern is addressed (`makes`, `helps`, etc.).

As Figure 6 illustrates, sometimes concerns play a role as a result of implications (instances of class Implication) of architecture decisions made externally to the ontology, such as a decision within another project or at the enterprise level. We normally incorporate these external decisions into our model and link the implications to our architecture decision. In addition, we find that an implication resulting from one architecture decision often affects other architecture decisions; that is, some implications become concerns. In practice, we simply model class Implication as a subclass of Concern. A better choice would be to define a formal Implication-Concern relation. The Implication-Asset Relationship class shown in Figure 6 defines the way an implication of an alternative affects architecture assets; it takes the values `creates`, `modifies`, `uses`, and `retires`. For example, an implication may create an instance of Component. Capturing this relation explicitly enables us to perform an impact analysis, such as determining the effect a given alternative has on assets.

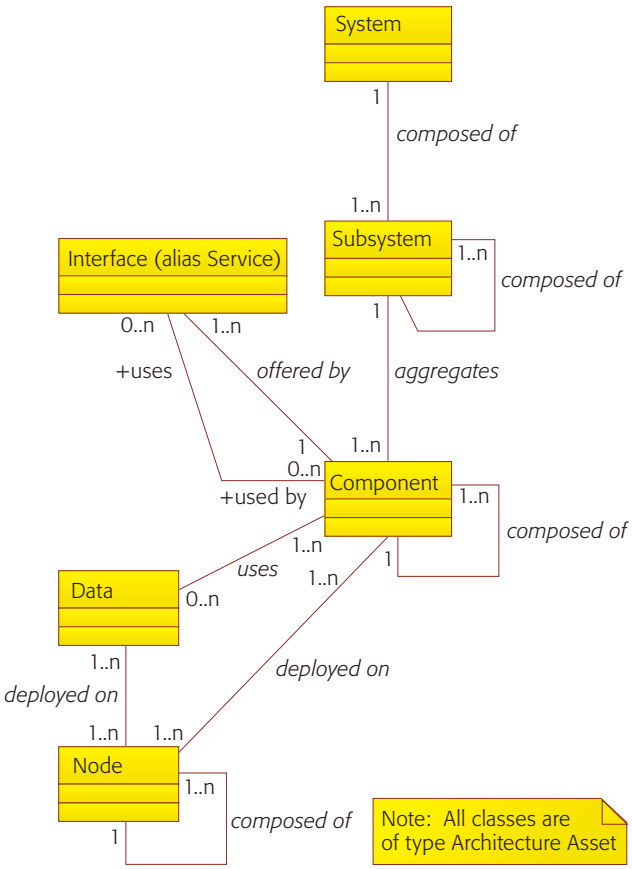

Figure 4
Architecture Asset segment of ontology

For our case study, the primary concern "offer customers a real-time credit-approval service" leads to our first decision: "Extend system B to include interactive credit-approval processing" (decision D01). The screen capture in ***Figure 7*** shows a view of the ontology tool when decision D01 and one of the alternatives are selected.

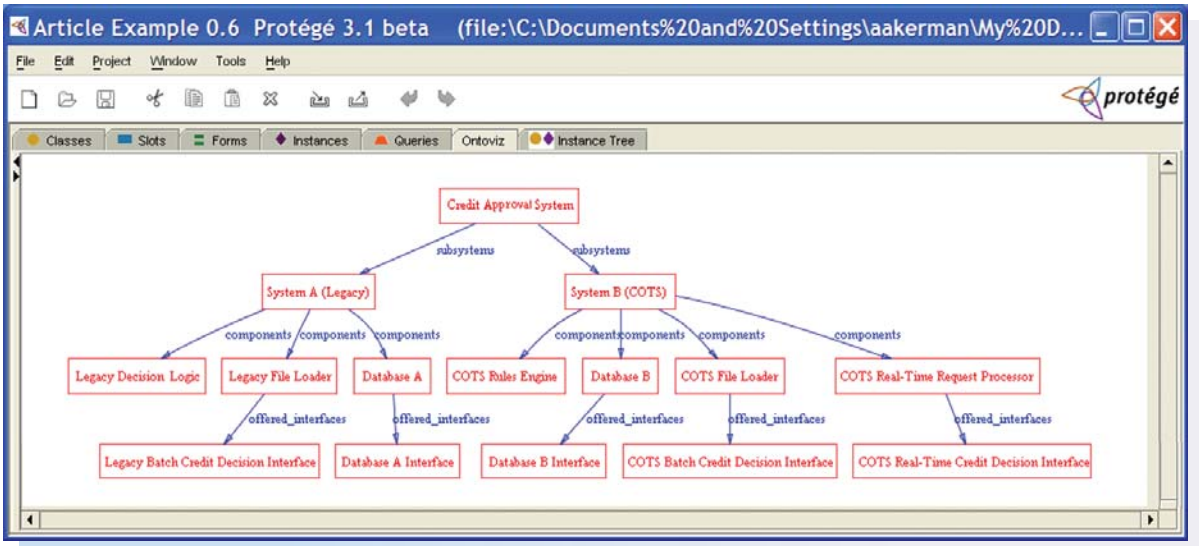

Figure 5
Protégé-generated view of credit-approval system

Table 1 describes the results obtained by applying an informal architecture-decision-based analysis to the same scenario as in our case study. The table includes entries such as the problem definition (issue), key assumptions (assumptions), alternatives being considered (positions), and the implications of selecting a particular alternative (implications). The loosely worded problem analy-

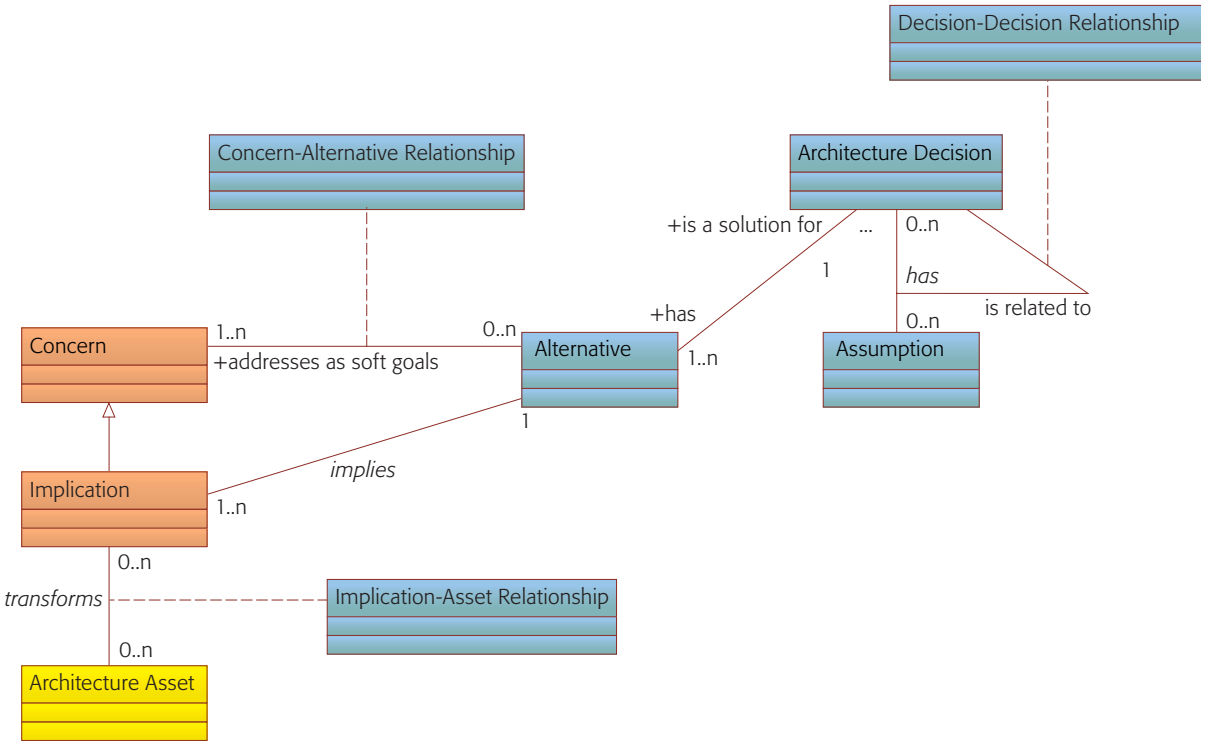

Figure 6
Architecture Decision segment of ontology

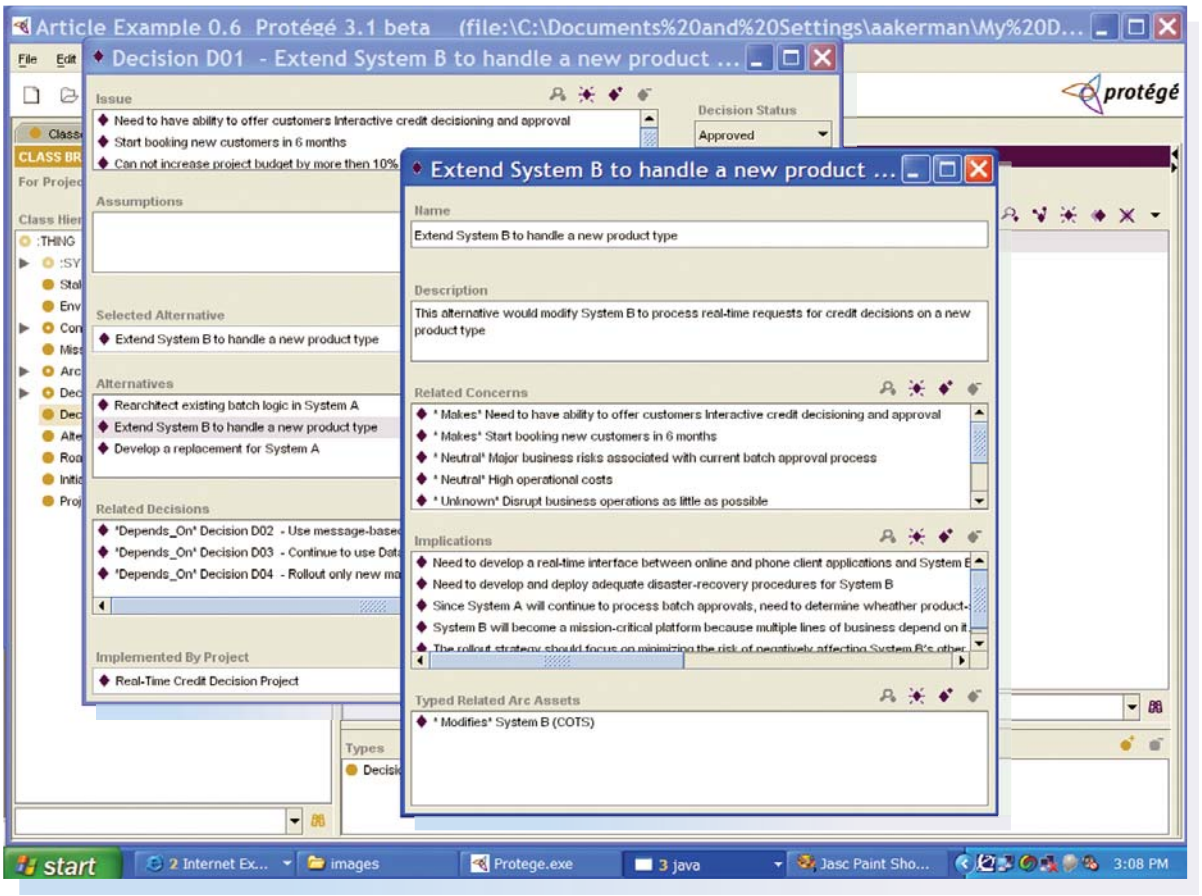

Figure 7
Decision DO1 and a selected alternative as viewed in Protégé

sis in Table 1 stands in contrast to the formal list of concerns in the Issue box of Figure 7. The items in the table listed under Assumptions turn out to be just additional concerns. Consequently, it is simpler to add these to the set of concerns and do away with the category Assumptions. This example illustrates that our ontology-driven approach yields more precise descriptions for architecture development.

Based on our understanding of the current environment, we create three alternatives (instances of Alternative). We evaluate these against all the relevant concerns by using a framework of softgoals evaluation.[15] The best alternative is the one with the most positive assessments (`makes` or `helps`). We also specify how each alternative affects one or more architecture assets. This is accomplished by using the connection from alternatives to implications to assets (see class Implication-Asset Relationship in Figure 6). This ensures that we understand how our systems change as a result of selecting one or another alternative. It also ensures that we capture all the appropriate implications, which may require separate architecture decisions. In our case, it is important to determine how clients interface with the new credit-approval system (system B), which database is used as master storage, and how system migration is handled.

Often an architecture decision creates more questions than it answers. In our example, a decision to use system B as a real-time credit-approval platform requires that we provide a solution for integration with clients (desktop and Internet), identify a system of records for the new product information, define a migration approach, and so forth. We document these issues as implications (instances of class Implication), which automatically allows us to associate these as concerns with the new architecture decisions (Implication is a subclass of Concern).

Table 1 Informal architecture-decision-based analysis of the credit-approval system

Issue	Current IT infrastructure does not support real-time approval processing for most financial offerings.
Decision	Extend system B to implement real-time approval processing. System B will provide real-time approval processing for all financial products currently handled by system B.
Status	Approved
Grouping	System structuring
Assumptions	• New capabilities will be delivered within six months. • Budget overrun will not exceed 10 percent. • Existing client applications will be used
Constraints	• None
Positions	• Redesign existing batch logic in system A • Extend system B to handle a new product type • Develop a replacement for system A
Argument	Extending system B to handle approval processing for all financial products will • reduce duplication of business logic, • allow all lines of business to use flexible workflow and rules engines to improve time to market for new products, • reduce maintenance costs and operational risks, and • have a solid chance of meeting project time lines, because the IT organization is already familiar with proposed technology.
Implications	Need to develop real-time interface between online and phone transactions to system B. System B will become a mission-critical platform for the business as several applications depend on it. There is a need to develop and deploy adequate disaster-recovery procedures. Rollout strategy should focus on minimizing adverse impact to other financial products supported by system B.

Should architecture decisions address all implications? We don't think so. We create architecture decisions only for implications that either have high implementation risk or are not addressed by existing decisions, company policies, or procedures; for example, there is an implication to make system B "mission critical." Because procedures already exist for transitioning a system to this status, we do not have to make an explicit decision about it.

We end up with the following eight additional architecture decisions:

1. Use a message-based middleware platform for real-time interfaces.
2. Continue to use the system-A database to store product-specific data.
3. Use a new platform for deploying new financial products exclusively.
4. Continue to populate the data warehouse from the system-A database.
5. Use XML as the message format.
6. Replace all batch interfaces.
7. Use API-based middleware for current clients.
8. Create interfaces between message-based and API-based middleware.

Only the first architecture decision ties directly to the set of concerns. It is interesting to observe that the additional eight decisions identified above address the implications of the first one. We should point out, however, that all these decisions do not address all concerns. Specifically, they do not address the high operational costs of running the IT infrastructure or reducing the risks associated with the current batch approval process. We would need to develop additional architecture decisions. In our experience, this is a typical problem with less precise methodologies for developing architectures. Use of ontologies ensures that the architect does not miss important stakeholder needs.

Conducting a gap analysis and producing the roadmap

Figure 8 shows the Roadmap segment of the ontology, which completes the working model. The

roadmap provides direction on migrating from the current system to the target system. It organizes the work as a number of instances of class Initiative and the relation `organized by`. These instances in turn are associated with instances of class Project through the relation `implements part of`. In other words, the work consists of a number of initiatives, each of which is carried out in a number of projects. An initiative generally covers a cohesive set of stakeholder concerns, such as improvements to environment stability, or time-to-market considerations. The projects put architecture decisions into practice within an overall plan by implementing their implications. The implementation of a particularly large or complex architecture decision may span multiple projects by allocating the implementation of a subset of its implications to separate projects. This relation enables a temporal analysis that provides a view of the architecture following the execution of a series of projects. It also enables impact analysis on architecture assets, for example, upon the cancelation of a project.

For simplicity, the implementation for this case study relies on a single instance of class Project, Real-Time Credit Decision Project (see Figure 7). Because this project has an implementation date, it is possible to see the architecture at any particular moment in the future (before and after implementation). In order to generate a target (future) architecture view we would have to link a project with the implications that it implements. The implications link with the architecture assets through the `creates`, `modifies`, `uses`, or `retires` relations. The target view hides retired assets, while showing the assets created, used, and modified. Note that although our model does contain all necessary information, our current ontology tool (Protégé) does not support automatic generation of such views.

Validating the architecture

Architectural reviews are the most common form of software-architecture validation. We employ several levels of reviews, such as executive reviews, peer reviews, and reviews by an architectural review council. The common approach to describing architecture is based on a collection of views (logical structure, physical deployment, runtime processes, etc.). The format and the specific content of the reviews differ depending on individual needs, background, and working styles. The content

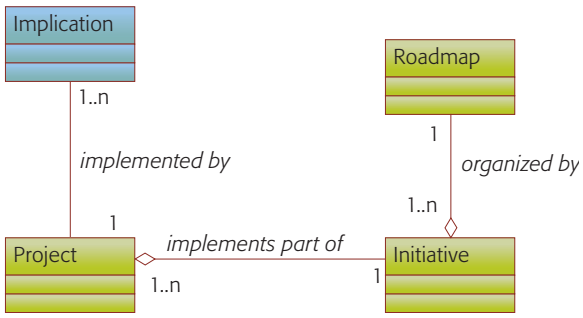

Figure 8
Roadmap segment of ontology

needed by all reviewers to validate the architecture corresponds to the four related ontology segments:

1. A high-level assessment by domain as to how the architecture meets stakeholder concerns
2. Key architecture decisions
3. Impact on architecture assets, such as legacy systems to be retired, new components to be implemented, and batch and real-time interfaces to be modified
4. A roadmap for migrating to the target architecture, depicted as a series of projects

The proposed ontology enables us to capture this content as we make architecture decisions. We are able to construct temporal views, which support incremental, agile, and ad hoc reviews. In addition, as the ontology is agnostic with respect to any specific architecture view, the information can easily support the construction of any set of views required by traditional view-based approaches.

DISCUSSION

Earlier we identified a number of possible benefits that this ontology-based approach offers beyond our original decision-driven architectural development approach. We examine now these possible benefits in the light of our experience from the case study:

- *Improved focus on what is important*—In our approach, we start with a prioritized set of concerns. These concerns lead us to a set of architecture decisions that address the concerns. The architecture decisions may generate additional architecture decisions by way of implications introducing additional concerns. We achieve architectural integrity when this iterative process

converges, and there are no more concerns to address.

- *Improved precision and clarity*—Our ontology relies on precisely defined concepts and relations rather than on free-form descriptions. A problem description is expressed as one or more concerns; implications are realized by transforming architecture assets; and the extent to which architecture decisions address concerns is evaluated through concern-alternative relationships.

- *Repository support*—It is possible to create a repository of architecture definitions by using one of the ontology development tools, many of which are available in open-source form. These tools support common technology standards like OWL (Web Ontology Language), XML, and XMI** (XML Metadata Interchange), which make them interoperable with modeling environments from IBM Rational and Borland Software Corporation.

- *Support for impact analysis*—An architectural ontology helps us establish relations between ontology elements (we refer to this property as traceability). Performing impact analysis is just a matter of creating the right queries. For example, if we change an architecture decision, we can see all the other architecture decisions depending on it by using the following SQL statement:

  SELECT D.DECISION_ID FROM DECISION D,
  XREF_DECISON_RELATED_DECISIONS D_D
  WHERE D.ID = D_D.ID2 AND
  D_D.ID1 = CHANGED_DECISION_ID.

  Note that this SQL query works with our custom relational database schema generated using data from the Protégé ontology file. This example of impact analysis is of the type "decision to decision." Analyses of the type "decision to concern" and "decision to architecture assets" can be similarly performed.

- *On demand view creation*—We generate views automatically as needed from the information in our repository (see Figure 5). In addition to diagrams, we also generate HTML-based views. These are similar to the standard architecture documentation, but because they are documented in HTML (Hypertext Markup Language), they are easier to navigate.

Our experience shows that applying our approach to real-life situations has some limitations, primarily due to our tool selection. These limitations can be addressed by selecting a different tool or by developing custom Protégé plug-ins.

- *Lack of support for reified relations*—Relations (between concepts) that are represented as classes with their own slots are referred to as *reified* relations. Class Concern-Alternative Relationship, for example, allows us to define how a particular alternative addresses a concern. Unfortunately, handling this type of class in Protégé is awkward because there is no way to establish a connection between an instance of this class and a relation. This means that when a relation is removed or modified, the class has to be deleted or manually modified.

- *Difficulties in generating standard architecture views*—Although there are many Protégé plug-ins for ontology visualization, the views they create are often busy and somewhat confusing. Configuring the tool to produce acceptable diagrams using these plug-ins is time consuming, and the configuration parameters cannot be saved. A capability is needed to create template-based views, such as the view found in IBM Rational SoDA*.

- *Inadequate querying tools*—It is easy to create simple queries in Protégé, such as finding all instances of Concern that have the value `critical` for the property `priority`. It is much more difficult to find, for example, the architecture asset that has the value `modified` for the property `relation type` in class Implication-Asset Relationship. There are a number of plug-ins that support the RDQL,[16] SPARQL,[17] ARQ,[18] and F-Logic[19] query languages. However, considerable skill is required to use these. Protégé can use any JDBC-compliant relational database to store the data. Because a database table is used to store an entire ontology definition, this approach is not designed for external querying. We prefer to export our ontology into a relational database in which each class is stored in a separate database table. This allows us to query ontology data by using standard SQL.

RELATED WORK

Defining an architecture metamodel for documentation purposes is well understood (see Booch's

metamodel[20] as an example), but, beyond the use of components and connectors, which is captured by most architecture description languages, limited research has been done in architectural ontology. Kyaruzi and van Katwijk[21] propose an ontology that contains nine architectural concepts: agent, arrangement, resource, product, location, directive, mechanism, event and calendar. This ontology is compelling in that it refines the components-and-connectors level and provides a more concrete vocabulary for software architects. In addition, it guides architectural focus by directing architects to confine their decisions to this set of concepts. However, we find the ontology lacks a clear understanding of how the elements relate to each other and how they map to existing architectural concepts. More important, we have found that many of these elements do not show up in practice. We suggest leveraging enterprise-modeling languages with their accompanying ontologies for describing architecture assets. In addition to the Architecture Description Standard from IBM, other examples are Enterprise Architecture Modeling Language from Infosys Technologies Limited,[22] the modeling language defined as part of the ArchiMate project,[23] and the SEAM modeling language (SEAM stands for Systemic Enterprise Architecture Methodology).[24]

Architecture decisions are the centerpiece of the ontology and of the decision-based process we use. Our ontology is derived from the REMAP (Representation and Maintenance of Process Knowledge)[25] and DRL (Decision Representation Language) metamodels,[26] and Kruchten's[14] ontology of software architecture decisions. We build on these results, add the concepts of assets and the roadmap, and pull them all together by making the connections between them explicit.

Several tools exist for managing design decisions, such as DREAM (Design Rationale Environment for Argumentation and Modeling),[27] SEURAT (Software Engineering Using RATionale),[28] Sysiphus,[29] and OSC.[30] Of these tools only Sysiphus provides an extendable modeling environment and user-defined views over the model. The focus on the underlying architectural ontology, versus documentation, distinguishes our approach from that addressed by Sysiphus.

CONCLUSION

We propose a focal shift in software architectural practice from diagrams, models and views to architecture decisions and use of ontology tools. We submit that this shift starts with the use of a common vocabulary, one that resonates with business and IT stakeholders.

We present the ontology as comprising four major concepts, Concern, Architecture Decision, Architecture Asset, and Roadmap, which are highly interrelated through a set of explicit relations. We advocate adding them as extensions to the IEEE recommended standards.[1]

We suggest creating reference ontologies, similar to the way design patterns have emerged. Weiss and Araujo[31] show patterns mapped to architectural issues and decisions, resources and qualities. A catalog of this work is a possible beginning of an enterprise reference ontology.

We acknowledge that each of the connected segments of our ontology needs further exploration. In order to promote standardization as well as tool development and reuse, experimentation with ontologies is needed. The SysML[32] requirements ontology, for example, can be leveraged to elaborate concerns. Enterprise modeling languages provide a starting point in defining an ontology for architecture assets. We should point out, however, that additional aspects, such as migration, conversion, deployment, and application disposition, would have to be included. Finally, every architecture must be implementable. The architecture roadmap segment of the ontology requires better integration with the broad area of enterprise portfolio management.

As practicing architects, we continue to use the described approach in our architecture development work. We do not claim that it will fit every organization, but it is proving to be very effective in ours. We are committed to refining our approach in the hope of improving the quality of architectures and the systems resulting from them.

CITED REFERENCES

1. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE STD 1471-2000, IEEE (2000).

2. J. Putman, *Architecting with RM-ODP*, Prentice Hall PTR, Upper Saddle River, NJ (2001), p. 834.
3. P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software* **12**, No. 6, 42–50 (1995).
4. P. Kruchten, *The Rational Unified Process: An Introduction*, Addison-Wesley, Reading, MA (2000), p. 298.
5. J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture," *IEEE Software* **22**, No. 2, 19–27 (2005).
6. M. Arseniev, "Enterprise Architecture Implementation: Practical Steps Using Open Source Tools," *College and University Machine Records Conference (CUMREC)*, (2004).
7. T. R. Gruber, "A Translation Approach to Portable Ontologies," *Knowledge Acquisition* **5**, pp. 199–220 (1993).
8. N. F. Noy and Deborah L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," Knowledge Systems Laboratory Technical Report, Stanford University, Stanford, CA (2001).
9. A. Akerman, J. Tyree, and L. Coglianese, "An Architecture Process for System Evolution," *Enterprise Architect Magazine* **2** (Spring 2004).
10. TOGAF 8 "Enterprise Edition," The Open Group, http://www.opengroup.org/togaf/.
11. "The Protégé Project," *Stanford Medical Informatics*, Stanford University School of Medicine (2005), http://protege.stanford.edu.
12. R. Youngs, D. Redmond-Pyle, P. Spaas, and E. Kahan, "A Standard for Architecture Description," *IBM Systems Journal* **38**, No. 1, 32–50 (1999).
13. B. Droge, "Methodische Integratie van Verschillende Service Disciplines in Projecten," *Landelijk Architectuur Congres (LAC2002)*, Zeist, Netherlands (November 2002), pp. 1–12. (in Dutch, "Methodical Integration of Different Service Disciplines in Projects," *Proceedings of the Nationwide Architecture Conference)*.
14. P. Kruchten, "A Taxonomy of Architecture Design Decisions in Software-Intensive Systems," *Second Groningen Workshop on Software Variability Management (SVM2004)*, December 2–3, 2004, Groningen, TheNetherlands, http://www.rug.nl/informatica/onderzoek/programmas/softwareEngineering/SVM2004.
15. L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Springer, New York (1999).
16. A. Seaborne, "Jena Tutorial: A Programmer's Introduction to RDQL," SourceForge.net (2002), http://jena.sourceforge.net/tutorial/RDQL/.
17. "SPARQL Query Language for RDF," E. Prud'hommeaux and A. Seaborne, Editors, *W3C Candidate Recommendation* (April 6, 2006), World Web Consortium, http://www.w3.org/TR/rdf-sparql-query/.
18. A. Seaborne, *ARQ-A SPARQL Processor for Jena* (2005), http://jena.hpl.hp.com/~afs/ARQ/.
19. "The XSB Programming System (Version 2.2)," Department of Computer Science, SUNY at Stony Brook (April 20, 2000), http://www.cs.sunysb.edu/~sbprolog/xsb-page.html.
20. G. Booch, "Software Archeology," *Rational User Conference* (2004), http://www.booch.com/architecture/blog/artifacts/Software%20Archeology.ppt.
21. John K. Kyaruzi and J. van Katwijk, "Beyond Components-Connections-Constraints: Dealing with Software Architecture Difficulties," *14th IEEE International Conference on Automated Software Engineering* October 12–15, 1999, Cocoa Beach, Florida, IEEE, New York (1999), pp. 235–242.
22. S. Sarkar and S. Thonse, "EAML—Architecture Modeling Language for Enterprise Applications," *Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-East 2004)*, IEEE, New York (2004), pp. 40–47.
23. H. Jonkers, B. Buuren, F. Arbab, F. de Boer, M. Bonsangue, H. Bosma, H. ter Doest, L. Groenewegen, J. G. Scholten, S. Hoppenbrouwers, M.-E. Iacob, W. Janssen, M. Lankhorst, D. van Leeuwen, E. Proper, A. Stam, L. van der Torre, and G. V. van Zanten, "Towards a Language for Coherent Enterprise Architecture Descriptions," *Proceedings of the Seventh International Enterprise Distributed Object Computing Conference (EDOC'03)*, September 16–19, 2003, Brisbane, Australia, IEEE, New York (2003), pp. 28–39.
24. L.-S. Lê and A. Wegmann, "Definition of an Object-Oriented Modeling Language for Enterprise Architecture," *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS'05)*, January 3–6, 2005, Big Island, HI, IEEE, New York (2005), pp. 222a.
25. B. Ramesh and V. Dhar, "Supporting Systems Development by Capturing Deliberations during Requirements Engineering," *IEEE Transactions on Software Engineering and Methodology* **18**, No. 6, 498–510 (June 1992).
26. J. Lee and K.-Y. Lai, "What's in the Design Rationale?" *Human-Computer Interactions* **6**, Nos. 3–4, 251–280 (1991).
27. X. Lacaze, "Design Rationale for Interactive Systems," Institut de Recherche en Informatique de Toulouse, France, http://liihs.irit.fr/lacaze/research.html.
28. J. E. Burge and D. C. Brown, "An Integrated Approach for Software Design Checking Using Design Rationale," *Proceedings of the Design, Computing and Cognition Conference*, July 19–21, 2004, Cambridge, MA (2004), http://web.cs.wpi.edu/~dcb/Papers/DCC-paper-04.pdf.
29. T. Wolf and A. H. Dutoit, "Sysiphus: Combining System Modeling with Collaboration and Rationale," http://pi.informatik.uni-siegen.de/stt/24_4/01_Fachgruppenberichte/11wolf.pdf.
30. G. Arango, L. Bruneau, J. Cloarec, and A. Feroldi, "A Tool Shell for Tracking Design Decisions," *IEEE Software* **8**, No. 2, 75–83 (1991).
31. I. Araujo and M. Weiss, "Linking Patterns and Non-Functional Requirements," *Proceedings of the Ninth Conference on Pattern Language of Programs (PLoP 2002)*, September 8–12, 2002, Monticello, Il, http://jerry.cs.uiuc.edu/~plop/plop2002/final/PatNFR.pdf.
32. D. W. Oliver, "Systems Engineering Conceptual Model (Draft 12)" (March 27, 2003), http://syseng.omg.org/SE_Conceptual%20Model/SE_Conceptual_Model.htm.

Accepted for publication December 26, 2005.
Published online October 16, 2006.

Art Akerman
Capital One Financial, 15000 Capital One Drive, Richmond, VA 23238 (art.akerman@capitalone.com). Mr. Akerman is a system architect at Capital One Financial and a practicing member of the World Wide Institute of Software Architects. His research interests include creation of formal education programs for software architects, communicating architecture

to the development community, and making architecture development more practical and less time consuming. He received a Master's degree in management of information technology from the University of Virginia.

Jeff Tyree
Capital One Financial, 15000 Capital One Drive, Richmond, VA 23238 (jeff.tyree@capitalone.com). Mr. Tyree is a solutions architect at Capital One Financial. His research interests include large-scale system design, system evolution processes, refactoring, and performance engineering. He received a Master's degree in mathematics from the University of Tennessee at Knoxville. ■