# Beyond the Zachman framework: Problem-oriented system architecture

B. Robertson-Dunn

*The year 2012 marks the twenty-fifth anniversary of "A framework for information systems architecture," written by John Zachman and published in the IBM Systems Journal. The first part of this paper reviews the Zachman and similar frameworks and concludes that there are a number of limitations in the framework approach when applied to today's technology environment and business problems. These include the inability of the problem owner to properly describe a solution, the partitioning approach, and the decision-making processes in the context of uncertainty and change. The second part of this paper analyzes today's problems and allocates them to one of three classifications: tame, complex, and wicked, depending on the degree of certainty and stability of knowledge and decisions in both the problem and the solution domains. The final part outlines an approach to problem-solving and architecture development using techniques borrowed from cybernetics and control theory. It proposes that partitioning should be determined by the nature of the problem and potential solutions; that feedback loops should be implemented in order to control the process; that the architect should work across the business problem and solution spaces; and that decisions should be related to business value.*

## Introduction

In 1987 the *IBM Systems Journal* published "A framework for information systems architecture" by John Zachman [1]. This was the first attempt at developing a structured approach to what has become known as enterprise architecture. Structured analysis to underpin software had been developed earlier, but John Zachman and others working at IBM on large-scale technology problems recognized that there is more to a system than just the software.

In this the twenty-fifth anniversary year of the publication of the Zachman framework, it is worth examining the changes in technology and systems that have occurred in that period and assessing the usefulness of the Zachman approach in light of current business problems, technologies, and systems. The Zachman framework was, at the time, a major step forward in understanding business information systems and provided an analytic basis for subsequent system development. The idea of architectural frameworks has been developed over intervening years and is now considered an essential part of the architecture and system development disciplines. The Zachman framework has been extended [2], and a number of other frameworks have been developed [3–5]. These have been reviewed and compared elsewhere [6–8].

This paper examines the Zachman and other frameworks in the context of decision-making in an environment of uncertainty and assesses their suitability for use in today's technology and business environment. A more dynamic and problem-oriented approach to architecture and system development, referred to as a problem-oriented system architecture (POSA), is proposed, one that leans heavily on concepts and techniques from cybernetics and control theory.

## The Zachman framework

The Zachman framework describes the structure of a system architecture and processes for its development. This

Digital Object Identifier: 10.1147/JRD.2012.2205633

framework is used for organizing, creating, and managing architectural assets to support the understanding and development of business information systems. In developing the framework, Zachman analyzed existing disciplines in order to determine ways in which to create structure and proposed his framework within which analysis and system specification can be organized:

> "In searching for an objective, independent basis upon which to develop a framework for information systems architecture, it seems only logical to look to the field of classical architecture itself. In so doing, it is possible to learn from the thousand or so years of experience that have been accumulated in that field. Definition of the deliverables, i.e., the work product, of a classical architect can lead to the specification of analogous information systems architectural products and, in so doing, can help to classify our concepts and specifications" [1].

This idea of learning from other disciplines is the first of a number of themes John Zachman developed in his framework.

The second is that a system will look different when viewed from different viewpoints and when developing architectural representations. The Zachman framework arranges these architectural representations into a matrix of perspectives and descriptions. Perspectives were defined in the original paper as the owner, the designer, and the builder. The descriptions specified were answers to the what, how, and where questions. Later versions also included who, when, and why.

The third theme is that "the architect's drawings are a transcription of the owner's perceptual requirements" [1]. In other words, the role of the architect in populating architecture assets is to work with the system owner, who "has in mind a product that will serve some purpose" [1] and who provides the requirements and makes decisions regarding changing or adding to the set of requirements.

The technology available to developers of business systems in 1987 was relatively limited. The user devices had little variability—mostly block-mode 3270-style terminals or personal computers before the days of graphical user interfaces and widespread networking. The Internet had not become ubiquitous, and technology was owned by the enterprise that also owned the business processes. Technology choices and hence decisions were limited; network parameters were mostly about location.

At that time, the information and communications technology (ICT) environment was relatively static. Initially, the Zachman framework was applied to large mainframes, which either came with or had limited choices as to databases and programming languages. The technology was based on architectures provided by a single vendor, so there was little opportunity to make technological or network decisions. Many of the business systems were implemented as batch processing.

Problems were internal to the business, and technology solutions were for the most part used only by employees of the business, to support that business. There was little or no external electronic communications, systems were tightly tied to predefined business processes, and most of the information was highly structured.

Business processes did not change very quickly. As Zachman said in "Zachman on the framework" [9],

> "It is my opinion that if you define the primitives relative to the Enterprise, they likely do not change appreciably as long as you stay in the same business."

In the years since 1987, other frameworks have been developed. Some are enterprise architecture frameworks [The Open Group Architecture Framework (TOGAF**), Federal Enterprise Architecture Framework (FEAF)], and others are aimed at system development (DoDAF). Details of these frameworks are available elsewhere [3–5], along with a number of comparisons [6–8].

## Analysis of frameworks

Analysis and experience show that there are a number of limitations in the framework approach to architecture and system development today. These are outlined in the following sections.

### *Problem formulation*

In the framework approach, solving the problem is deemed to be the responsibility of the business system owner. Although the framework architect works with the system owner, it is in order to identify solution requirements, not to question whether the problem has been formulated or solved appropriately.

Frameworks rely on the business system owner being able to define the requirements of the solution. The assumptions behind this technique are highly questionable. Edward Yourdon, in *Rise and Resurrection of the American Programmer* in 1997 [10], observed that it is not possible to identify user requirements completely and accurately. Users rarely know what they want and are even less able to articulate it. Even if they can, the requirements are highly likely to change during the conduct of the project. The only things that have changed since Edward Yourdon made these observations are that the business and technology environments are far more complex, uncertain, and changeable, the consequence being that it has become even more difficult for users to solve their problems and then identify and communicate requirements. This lesson has not been learned by the architecture and system development communities.

## Partitioning

A feature of the framework approach is that the architecture space for understanding the requirements and developing a solution is partitioned. This is an aid to analysis and specification of the solution; however, there are a number of dangers.

There is no reason why partitioning in the problem domain should be the same as that in the solution domain. Different understandings are required and are for different purposes. It is also possible that different parts of an organization might require different types of partitioning. An enterprise such as the Department of Defense has a wide range of organizational units, each with significantly different business problems and constraints on their solutions.

Likewise, a service delivery area of government, with its need for transaction-based systems, potentially has different business problems from those of regulatory or policy departments, which are mainly information-based. If the optimal solution is a product from a vendor, then it is possible that the architecture within that product is irrelevant or could be incompatible with the partitioning developed by the system architect. In addition, a services-oriented architecture or cloud technology approach might cause problems when related to a partitioning style imposed by a rigid framework.

A further disadvantage of partitioning is that most nonfunctional aspects of a system work across the whole system. These aspects include security, performance, availability, resilience, business continuity, and backup. Some of these can be very difficult in a nonstop, distributed system, and special attention must be paid to them in the early phases of the system architecture development.

The Zachman framework explicitly excludes these issues, and so do most of the other frameworks. Partitioning, and allocating aspects of architecture and system development to different teams based on this partitioning could result in nonfunctional requirements not being met for the whole system.

Roger Sessions, in "A Better Path to Enterprise Architectures" [11], recognizes the problems of partitioning in a sequential approach. Sessions proposes a partitioned-iteration technique, which is based on dividing the total system into smaller subsystems and building each one in turn. This may work for some simple problems and solutions; however, it is still based on requirements not the problem. It does not address the weaknesses of partitioning. It simply moves them to different phases of the development, that is, to the implementation and integration stages. There is a real danger that decisions made during the development of one subsystem may not be valid or appropriate for another subsystem, either at the time they are made or at the time subsequent subsystems are developed.

## Decision-making, uncertainty, and change

The typical framework approach is to analyze the current environment, develop a target architecture, and then initiate projects to close the gap. This presupposes that a stable long-term target architecture can be developed, in light of knowledge gained during requirements analysis. It also assumes that the target architecture will remain stable over time and as the system development process proceeds. These are dangerous assumptions.

Decision-making in the architecture and early system development process usually occurs in an environment of uncertainty and/or lack of knowledge. Most of the frameworks assume that there is a sequential process of requirement identification permitting more detailed decisions to be made as the project progresses. However, the sequential nature of the framework approach does not easily accommodate uncertainty and change. Disturbances in the environment and in the problem space are not explicitly accommodated in any of the frameworks.

## Consequential problems

The framework approach is solution-oriented. The requirements are specified, and a solution is developed. However, all solutions either change the problem space or create problems of their own. By definition, a solution solves a problem, which is an intended change in the problem space. What is not recognized in the framework approach is that any solution will create new, consequential problems. Some might be serious enough to have an impact on the value of solving the problem, whereas others might change the nature of the problem. These problems are not identified as part of the framework process and so are not resolved.

## The value of architecture

In the framework approach, especially those concerned with enterprise architecture, it is claimed that there are benefits and advantages that accrue from developing an architecture. However, these benefits may or may not have value to a particular business. In order to assess a particular benefit, it is necessary to identify whether it is of value to that enterprise. A business operating in a highly competitive consumer environment might value flexibility and time to market very highly and a long-term roadmap far less so. A government department may decide that, for it, the reverse is more appropriate, because it is interested more in stability and certainty than speed of system development. An organization that has information systems that support a major sporting event such as the Olympics sets high priorities for performance, reliability, and availability, and these are probably only for the duration of the event. It puts a lower value on flexibility, maintainability, and other longer-term factors.

In general, an architecture initiative can best be justified to the business not in terms of benefits, but in terms of the value of the business problems it will solve or be instrumental in solving. Conversely, if an architecture initiative does not solve at least one business problem, the question "What is the value of applying resources to it?" needs to be asked. The business value of a solution exists in the problem it solves.

## Where are we now?

The limitations in the framework approach are compounded by the nature of the technology available to businesses today and the types of business problems to which technology is now being applied.

### *Technology today*

Technology has become pervasive and changes rapidly, even at levels that can affect fundamental characteristics of systems. Effective bandwidths and capabilities change, sometimes in contradictory ways. For example, the spread of small mobile devices has meant that the user's viewable area is restricted and network connectivity can be limited and sometimes unreliable. These have a negative impact on end-to-end system performance. At the same time, wide-screen, high-definition television and high-speed broadband Internet access have become widely available. This has opened up the potential for a whole new range of applications and business opportunities. The system requirements for each of these situations are different but may both need to be accommodated.

Systems today are characterized by a large number of unknowns and uncertainties, a lack of control over many of the users, and complex interactions with other systems. Many applications involve the general public, whose behavior can be unpredictable and who often have irrational likes and dislikes. Technology is more complex. Systems are more complex. Problems are becoming more difficult to solve. There is more uncertainty, there are more complex relationships, and generally change occurs more rapidly.

### *Business today*

The environments in which organizations operate today are often more uncertain, even hostile, as enterprises actively compete with one another. Today's businesses are tightly integrated into a larger context. Communications external to the enterprise are the norm. Most, if not all, large businesses cannot exist without their information systems. Changes in technology are driving major changes in the business and business processes. They are enabling different business models, changing the way existing businesses operate, and introducing new types of business. Many businesses have extensive presence on the Internet, and many businesses exist only in virtual space.

New business models are being developed, both in terms of what they do and how they do it. The publishing industry provides a rich set of examples of change. Physical book sellers have been affected by new technologies. Amazon now sells books online. It has no customer-facing facility, only a warehouse and delivery capabilities; deliveries come in bulk, and go out in smaller lots. In addition, many books do not exist in physical form. They exist as e-books and are read by users with a variety of electronic devices.

All of these changes and threats to the old book seller's business are the result of new and innovative information systems. Information systems have changed the way the old industry works while enabling a new industry with little in common with the old business.

## Problem-oriented system architecture

"Successful problem solving requires finding the right solution to the right problem. We fail more often because we solve the wrong problem than because we get the wrong solution to the right problem" [12]. The problem-oriented system architecture is an attempt to correct this failure when applied to architecture and system development.

### *The nature of system architecture*

I believe that the development of system architecture is a technical art, not a methodology. Methods and frameworks may help organize and communicate the thoughts and decisions developed through the architecture process, but they are neither necessary nor sufficient to achieve an optimal outcome.

I believe that the system architecture discipline has not kept pace with changes in technology and business over the past 25 years. The techniques based on identification of requirements, decomposition into independent areas, the specification of static, target architectures, and sequential development processes are not able to cope with rapid change and tight relationships between business and technology. There are whole classes of new business problems that the framework approach cannot cope with. The POSA outlined in this paper attempts to address the limitations in the framework approach identified earlier.

Underpinning the POSA is the concept of an architect being the bridge between a business problem and its solution. The architect must be able to understand the problem and assist in the development of potential solutions. It is unlikely that the architect will know enough or have authority unilaterally to make decisions in either domain. As such, the role of the architect is that of a translator between domains and a facilitator of decision making. The key role of a POSA architect is to operate in multiple domains, acting as a domain translator between the businesses problem and the solution, and facilitator for problem solving. The architect also needs to look outside the problem-solution space in order to identify potential disturbances that may affect the development project.

### *The problem-oriented system architecture approach*

POSA learns and borrows from cybernetics and control theory, disciplines that facilitate the understanding and control of dynamic processes [13, 14]. Some work has been done recently by Terry Hill of NASA to apply control theory to project management [15]. The POSA is a similar attempt to apply the control theory to architecture and system development. It should be noted that most of NASA's problems are engineering centric and fall into the complex class defined below. The problem is usually well defined, but what is not clear is the nature of the solution and how it is developed.

Much can also be learned from nonlinear, self-organizing, complex, adaptive systems; however, such lessons are beyond the scope of this article. This paper concentrates on the use of proper partitioning, feedback, predictive control systems, and optimization of outcome against potentially changing objective functions. POSA treats architecture and system development as a dynamic process that has as its desired outcome a solution to a problem that, when solved, is of value to the business. The solution may be implemented as a strategy and/or architecture, within which further development can occur, or as a specific system.

POSA is an iterative process of identify goal, formulate problem, predict one or more solutions, measure solutions against problem, measure problem against goal. The iterations either stop or become less frequent when an optimal solution has been selected and moves into implementation. The iterations need to continue to ensure that subsequent decisions do not change the solution or to ensure that the goal or problem has not changed, requiring modifications to the solution. The iterations move through various phases, recognizing that the basis on which previous decisions were made may change. At some point it might be acceptable to assume that the problem has been identified and remains constant. At another point, the assumption that the requirements will remain stable and constant could be safely made. In order to reduce risk, however, it is necessary to check that these assumptions are valid. The architect needs to control the POSA process, the system cannot control itself.

## *Classes of problems and solutions*

The first task facing the architect is to determine the best approach to solving a business problem. This requires understanding the nature of, and relationship between, the business goal, the business problem, and potential solutions. I allocate the business problems facing enterprise and system architects to one of three classes: tame, complex, and wicked. The classes are defined according to the degree of knowledge and certainty in the problem and solution domains. This is not a new classification scheme, it was developed by Nancy Roberts [16] in 2000 in the context of problem solving in the public sector.

**Tame problems**—The characteristics of a tame problem are that the problem is well understood and the solution is well defined. The solution to a tame problem does not significantly disturb the solution space. It must be possible for the problem to be analyzed and understood and for a set of criteria to be established in order to identify and define when it is solved. In addition, when implemented, the solution should not create additional problems that diminish the value of solving the original problem.

A sequential methodology can be applied in the development of a solution to a tame problem in the knowledge that over time, progress can be made without the need for re-work. Decomposition of the solution development should be possible without the risk of introducing unknown, critical, or unmanageable dependencies.

**Complex problems**—The characteristics of a complex problem are that the problem is well understood, but the solution is unclear or uncertain. The solution itself will probably have a significant impact on aspects of the solution space. The problem can be analyzed and understood and a set of criteria can be established in order to identify and define when it is solved.

Although it is clear what the solution to a complex problem must do and the criteria against which it is to be measured are specified, it is not known what the solution should be, how it can be implemented, and what the consequences of implementing the solution will be. In addition, there may be requirements of the solution to a complex problem that are not derivable from analysis of the problem but that arise because of the solution itself. This is a common situation in information systems where solving the problem requires particular application functionality. However, the application needs to run in a technology environment that supports nonfunctional requirements such as availability and performance. The environment required to support nonfunctional requirements both enables and constrains application functionality. It also has key relationships with other areas such as cost and ongoing support and maintenance. Other examples include when the solution needs to be integrated into an existing environment, when new technology is being utilized, and when the project team has not encountered this type of problem or solution previously.

It may also be that although the problem criteria are clearly stated, some of the requirements may be mutually exclusive. One solution might be small, inexpensive, and quick to implement, whereas another one might be large, complex, expensive, and time consuming to implement; however, a large, complex, inexpensive, and quick-to-implement solution is unlikely to be achievable.

**Wicked problems**—The characteristics of a wicked problem are that the problem is unclear, the solution is unclear, and the solution has an impact on the problem itself. The solution significantly disturbs the problem space.

The term "wicked" problem was originally coined by Horst Rittel [17] in the context of a general theory of planning and was applied to problems that had a high degree of social complexity.

A wicked problem is difficult to model and understand. Entities and concepts can be difficult to define and their relationships may be unclear or change radically as events occur or over time. Selecting the best way to approach the problem becomes a problem itself. A significant challenge in trying to solve a wicked problem is that it is difficult to confirm that a proposed solution will fully or adequately solve a wicked problem. In addition, what compounds the situation is that when a particular solution is considered or implemented, the problem itself can change.

There are usually many consequences to implementing a solution. Managing these consequences can lead to new wicked problems. Sometimes the solution appears to be simple, but the wickedness lies in the difficulty of implementation. A wicked problem cannot be partitioned for analysis, a problem cannot be disconnected from its potential solutions, and the solution cannot be partitioned for implementation without major risks to implementation and nature of the problem.

In the context of information systems, it is useful to assume that if there are a significant number or variety of users and/or stakeholders, especially outside the direct control of the business, then the behavior of those users may change in ways that have an impact on the problem and hence the solution options. This, alone, is sufficient to make the problem wicked.

### Classifying a specific problem

The classification of a problem into one of the three classes is dependent not only on the problem itself, but also on the skills, competencies, and experiences of the people solving the problem. Incompetent or inexperienced practitioners can turn what should be a tame problem into a wicked problem.

Every problem should be considered, at least initially, as unique. Even though the problem and solution look the same as others experienced in the past, there will always be differences. The circumstances and context could be different, and the team delivering the architecture and system development is unlikely to be exactly the same; even if it is the same team, it will be a new experience.

## Applying principles of cybernetics and control theory

### Partitioning

A major feature of cybernetics and control theory is modeling to understand problems and to develop solution options. Partitioning is a process by which different aspects of a problem can be modeled from a variety of perspectives. Understanding a complex or wicked problem cannot be done by looking at its constituent parts. The consequences of inter-relationships and emergent behaviors are only visible when viewing the problem as a whole.

Cybernetics and control theory recognize that complexity and wickedness arise because of their nonlinear characteristics and that nonlinear systems cannot be understood by simplifying, by partitioning, or by any other mechanism without destroying the relationship between the understanding and the reality of the problem. The concept of "taming" a wicked problem is fatally flawed. Partitioning a solution into subsystems may help system development, but unless the solution is recombined and its behavior measured as a whole, there is no guarantee that the total system will work when finally implemented.

A system architecture is a set of partitioned models. Control theory would suggest that the approach to partitioning and modeling be tailored to meet the needs of the problem and solution. Unlike the framework approach, which predefines the models, POSA proposes that a degree of analysis of the problem and potential solutions be undertaken before decisions about partitioning and modeling are made. This is because the modeling approach can impose severe restrictions on subsequent phases of the architecture and development processes. Different partitioning and modeling approaches may be appropriate for the problem domain and the solution domain.

It is advisable to first understand the relationships and interdependencies that are created when partitioning a particular problem and then manage the difficulties created by this process. For example, a small user device imposes restrictions on the user interface, which can have an impact on the behavior of the application and the data flows between the different components of the system. This understanding of relationships should be the role of the architect, empowered by the techniques of feedback and comparisons.

### Feedback

POSA makes significant use of the principle of negative feedback. This is a technique that compares the input to a process with the output of that process. Corrective action is taken if there is a discrepancy. This is a very powerful controlling technique and is used in many control systems. By treating strategy, architecture, and development activities as a dynamic process and applying feedback techniques to that process, there is an increased likelihood that an optimal outcome will be achieved.

In order to establish an effective feedback loop, there are a number of issues that need to be addressed. In order to properly connect the problem space with the architecture and system development process, it is essential that the problem itself be part of the feedback loop. The primary reason being that, in the POSA, the problem is considered a dependent variable. What actually matters is the goal that the business is trying to achieve and the value derived from the problem being solved. The problem is dependent on the business goal. Bringing the problem into the feedback loop has a number of effects. First, the business goals associated with the problem are subject to analysis and scrutiny. In other words, the problem itself is questioned to ensure that its solution still achieves the business goals. Second, the behavior of the predicted solution is compared with the problem, not the requirements. This means that external disturbances, consequential problems, and uncertainty and change will automatically be addressed as part of the process. If the predicted solution does not solve the problem and solving the problem does not meet the goals, some form of corrective action will be taken.

Simple problems do not need advanced feedback; the process can be considered an open loop, with a degree of project monitoring. Feedback in the case of complex and wicked problems needs to be more advanced. The difference between a complex and a wicked problem lies in certainties and relationships in the problem space. The feedback loop around a wicked problem needs to have strong links to all the stakeholders in both the problem and the solution space. A complex problem probably has fewer stakeholders in the problem domain.

The problem-solving and solution development process gradually increases its scope as a project advances. Initially, it generates solutions options; the feedback process converges on a preferred, predicted solution that is followed by design and construction. The feedback process continues to operate, concentrating on ensuring that subsequent decision making continues to lead to a solution that solves the problem.

### *Comparisons*

It is a fundament tenet of control theory that if something cannot be measured, it cannot be controlled. Measurement can take one of two forms, counting or comparison. In the context of system architecture, what needs to be measured is the solution with respect to the problem that needs to be solved. In a project that is based upon the framework approach, measurements are usually only of the counting type, those of time and cost. If the solution is ever measured, it is usually compared with requirements, not with the business goals or the problem that needs solving.

However, in architecture and system development, measurements and comparisons are not easy. In industrial control systems, the output can usually be compared in some direct way with the input signal. In the case of architecture and system development, the output at any one time is not the intended solution but a representation of the solution. In fact any architecture or design artifact is best considered to be a prediction or estimate.

Unfortunately, a predicted solution cannot be directly compared with a problem. The problem and its solution are different things. It is necessary, therefore, to translate from the solution domain back into the problem domain. Part of this translation back into the problem domain should include a recombination of the components that have been introduced because of partitioning into a complete system. Another is to identify the behavior of the predicted solution. These activities are defined as "domain translation" and are a major role of the architect. All these concepts are integrated into the POSA, which is shown in **Figure 1**.

## Case studies

In 2008, Sir Peter Gershon conducted a review of Australian Government Information and Communication Technology (ICT) [18]. As part of that review, he recommended that the Australian Government Information Office (AGIMO) develop a whole government data center strategy and predicted that costs of about $AUD1 billion over a period of 15 years could be avoided. Subsequently, AGIMO initiated a project to develop a data center strategy. I joined the project team in July 2009 as lead IT architect and strategist.

It was determined that the goal of cost avoidance of $AUD1 billion over 15 years could best be achieved by answering the question "What can AGIMO do, at the whole of government level, that agencies cannot do themselves?" This statement became the problem to be addressed by the strategy.

The problem was identified very early on as a wicked problem. There were multiple stakeholders who were likely to exhibit unpredictable behaviors. Most of the government agency stakeholders had different objectives and concerns from those of AGIMO, a central agency. Knowing whether a particular solution would solve the problem was also uncertain. The scope of the problem included all of the federal government's data centers and the systems they hosted. This strategy phase addressed the structure and relationships of the complete set of data centers. Subsequent phases addressed issues of the systems within data centers.

Over time, all the principles of the POSA were applied to the strategy development. After an initial analysis, the problem was partitioned into data center demand from government agencies, data center supply, data center technology, and potential solutions. The solution space included procurement issues, migration of agency systems, and a range of potential technical and commercial initiatives.

A decision-making forum was created comprising representatives from various agencies who could assign business value to the various aspects of the problem being solved. My role as enterprise architect was to lead the development of potential architectures and solutions, identify

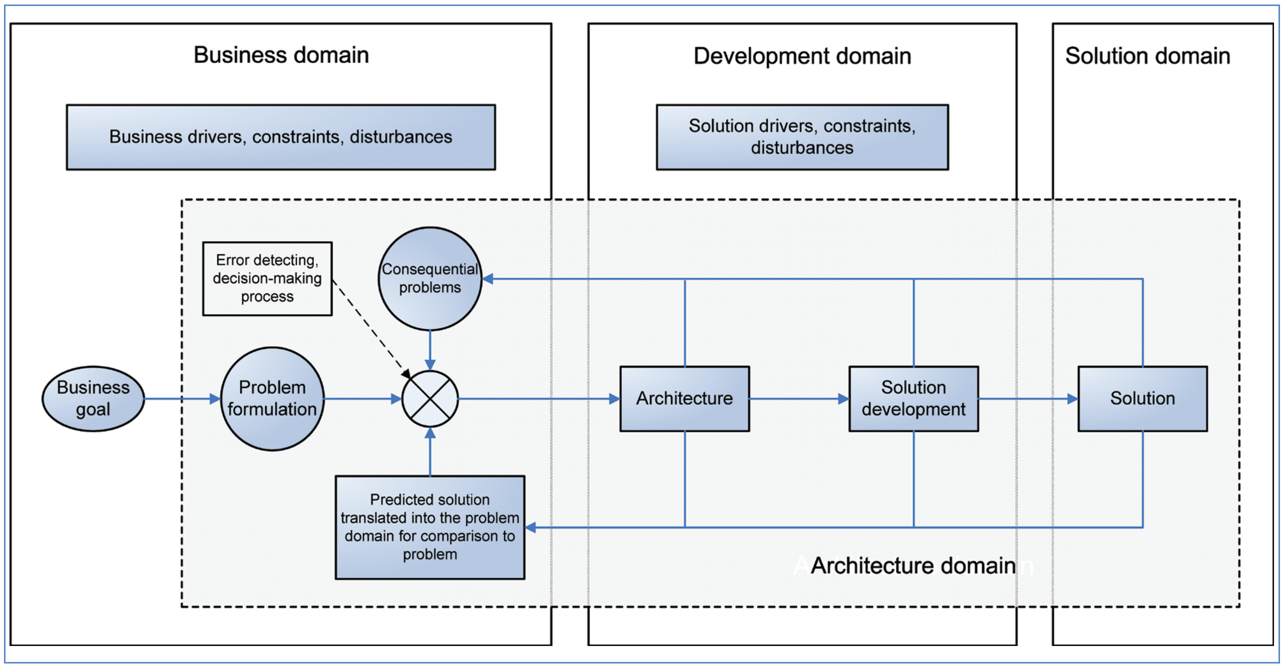
Business domain
Development domain
Solution domain
Business drivers, constraints, disturbances
Solution drivers, constraints, disturbances
Error detecting, decision-making process
Consequential problems
Business goal
Problem formulation
Architecture
Solution development
Solution
Predicted solution translated into the problem domain for comparison to problem
Architecture domain

Figure 1

The problem-oriented system architecture.

consequential problems, analyze the solutions, and translate them into forms that facilitated comparisons against each other and against the problem.

One of the difficulties the project team faced was in measuring the existing data center environment and its key characteristics. This was recognized during the strategy development and became a significant component of the final strategy. Over time, agencies will be required to report on an increasingly more comprehensive set of parameters.

The strategy was accepted by the Australian Government on March 22, 2010 [19], and a project was initiated to implement the strategy. The initial achievements were focused on coordinated procurement of data center facilities and data center migration capabilities [20]. In July 2011, AGIMO commenced the implementation of another phase of the strategy that is aimed at optimizing the technology that is housed in government data centers [21]. I am currently the lead IT architect on this project, which will, in part, achieve its goals through standardizing and consolidating ICT services by use of cloud services and associated technologies.

This development phase of the strategy was structured differently from the strategy itself; however, it also follows the approach of the POSA. My role in this phase is ongoing and is similar to that of the strategy development. The partitioning in this phase is driven by the need to comply with government procurement rules while also promoting competition in the Australian ICT marketplace and assisting government agencies to achieve value for money in the shortest time possible. The domain translation aspect is much more closely allied to technology and architecture than in the strategy development.

It should be noted that many of the activities undertaken in these projects were no different from those using any other approach. What was different from a requirements-driven approach was the role of the architect, supported by the structure of the project, the nature of the feedback, and the high priority given to understanding the problem, all of which are fundamental aspects of the POSA.

## Conclusion

This paper has described an approach that integrates problem-solving and solution development such that the whole process is explicitly aimed at the business and the problems that need to be addressed. Architecture and system development disciplines have not kept pace with the range of problems facing businesses and government today. Technology is more complex and capabilities change at a far more rapid pace than in the past. The relationships between business, business systems, and technology are more pervasive than ever.

Business problem owners generally do not have the training, skills, or experience in solving complex or wicked problems. Their perceptions of solutions, as specified through requirements, only partially describe a solution and how it is to be assessed. They do not describe their problem; at best, they describe symptoms. User-specified requirements are an unreliable place to start when developing today's business systems.

This approach has been termed problem-oriented system architecture because it moves away from the requirements/solution approach embedded in traditional frameworks to a wider scope of business value, business problems, solutions, and the problems that arise when implementing these solutions.

The problem-oriented approach is not directed at any particular architecture domain, such as system architecture or enterprise architecture. The POSA is fundamentally a problem-solving process underpinned by control theory, one that can be adopted for all types and scopes of business problems and architecture development.

The concepts and ideas of a POSA presented in this paper need to be more fully developed into a wider body of work such that more complex and wicked problems are better solved. It should result in a better track record of the IT industry in delivering large-scale system projects to business and government.

## Acknowledgments
I thank the Australian Government Information Management Office for allowing the use of the data center strategy as a case study in this paper. I also acknowledge the help and assistance given by David Oram (a former IBM Systems Engineer) who was the Project Director for the development of the strategy and who also provided highly valuable comments and critiques during the preparation of this paper.

## References
1. J. A. Zachman, "A framework for information systems architecture," *IBM Syst. J.*, vol. 26, no. 3, pp. 276–292, 1987.
2. J. F. Sowa and J. A. Zachman, "Extending and formalizing the framework for information systems architecture," *IBM Syst. J.*, vol. 31, no. 3, pp. 590–616, 1992.
3. TOGAF, Introduction, 2009. [Online]. Available: http://www.togaf.info/togaf9/chap01.html
4. The DoDAF Architecture Framework, version 2.02. [Online]. Available: http://www.eng.auburn.edu/~hamilton/security/DODAF/DoDAF_v2-02_web.pdf
5. *A Practical Guide to Federal Enterprise Architecture by the CIO Council,* version 1.0. [Online]. Available: www.gao.gov/bestpractices/bpeaguide.pdf
6. R. Sessions, *Comparison of the Top Four Enterprise Architecture Methodologies*. [Online]. Available: http://www.objectwatch.com/white_papers.htm#4EA
7. J. Schekkerman, *A Comparative Survey of Enterprise Architecture Frameworks*. [Online]. Available: https://docs.google.com/viewer?a=v&q=cache:aTRsMBNMYWwJ:www.enterprise-architecture.info/Images/Documents/Comparative_Survey_of_EA_Frameworks.pps+A+Comparative+Survey+of+Enterprise+Architecture+Frameworks&hl=en&gl=us&pid=bl&srcid=ADGEESiK3JhwjsUebmmPAmqkrBKozBrXAB6ics7vf27UpJX6ZpSL7eXLI4_t4wdBKHdsZDvAi6CosdsaYh5NvlzTjEKg6RqIWOJujzZLk1OP2GwkdDYk1XH2tk1FCNyla4F47u-WiOUY&sig=AHIEtbRecl0G7GZegDZutAbQK7tcKgIxjA
8. S. R. Susarapu and E. White Baker, "Analyzing enterprise architecture integration at the DHS using the Zachman framework," in *Proc. Southern Assoc. Inf. Syst. Conf.*, Atlantic Beach, FL, 2007, vol. 10, pp. 172–177. [Online]. Available: http://sais.aisnet.org/2007/SAIS07-40 Susarapu-Baker.pdf
9. J. A. Zachman, *Zachman on the Framework*. [Online]. Available: http://xpertaml.com/backup/ABS Development (Martin)/Methodologies/ZIFA/ZIFA09.pdf
10. E. Yourdon, *Rise & Resurrection of the American Programmer*, Englewood Cliffs, NJ: Yourdon Press, 1997.
11. R. Sessions, *A Better Path to Enterprise Architectures*. [Online]. Available: http://www.objectwatch.com/whitepapers/ABetterPath-Final.pdf
12. R. L. Ackoff, *Redesigning the Future: A Systems Approach to Societal Problems*. New York: Wiley, 1974.
13. N. Weiner, *The Human Use of Human Beings*. Methuen, MA: Riverside Press, 1950.
14. J. Doyle, B. Francis, and A. Tannenbaum, *Feedback Control Theory*. New York: Macmillan, 1990.
15. T. R. Hill, "Project Management Using Modern Guidance, Navigation and Control Theory," in *Proc. IEEE Aerosp. Conf.*, Big Sky, MT, 2011, pp. 1–14.
16. N. Roberts, "Wicked problems and network approaches to resolution," *Int. Public Manage. Rev.*, vol. 1, no. 1, pp. 1–19, 2000.
17. H. Rittel and M. Webber, "Dilemmas in a general theory of planning," *Policy Sci.*, vol. 4, pp. 155–169, 1973. [Online]. Available: http://www.thestudiony.com/ed/bfa/Rittel+Webber+Dilemmas.pdf
18. Australian Government ICT Review 2008 (Gershon Review). [Online]. Available: http://www.finance.gov.au/publications/ict-review/index.html
19. Australian Government Data Centre Strategy. [Online]. Available: http://www.finance.gov.au/e-government/infrastructure/australian-government-data-centre-strategy.html
20. Shed Services, Initiating, Instituting and Implementing the Australian Government Data Centre Strategy 2010–2025. [Online]. Available: http://agimo.govspace.gov.au/files/2011/03/Gartner_IODC_March_2011_-_Shed_Services.pdf
21. Australian Government's Data Centre as a Service Initiative. [Online]. Available: http://agimo.govspace.gov.au/2011/10/20/industry-consultation-data-centre-as-a-service/

*Received February 15, 2012; accepted for publication April 11, 2012*

**Bernard Robertson-Dunn** *Nicholls, ACT 2913 Australia (brd@iimetro.com.au).* Dr. Robertson-Dunn has a B.Eng. degree in electrical and electronic engineering, an M.Eng. and a Ph.D. degree, both in control engineering, all from Sheffield University in the United Kingdom. His Ph.D. thesis was on dynamic modeling of the electrical activity in the human small intestine. He has undertaken many and varied modeling assignments including anti-submarine weapons systems, financial modeling, corporate restructuring, information systems development, business process engineering, enterprise architecture, and information and communications technology (ICT) strategy. Since 1991, he has worked in Canberra, Australia, mainly on Australian government projects in the areas of defense, health, and a range of whole-of-government ICT initiatives. His primary focus has been on problem definition and solution strategies. He spent nearly nine years with IBM in Canberra in the IBM Government Business Unit where he achieved IT architect certification. At IBM, he undertook a range of roles, including enterprise architect and IT architect on several large Australian government ICT outsourcing projects as well as for a major Australian Airline. He is currently an independent consultant working for the Australian Government Information Management Office on strategies for the adoption of cloud computing. He is a member of Engineers Australia, the Australian Computer Society, the Institution of Engineering and Technology (United Kingdom), and the Institute of Electrical and Electronics Engineers.