An Integrated Modeling Approach to Enterprise Systems Architecture

By Santonu Sarkar & Srinivas Thonse

Existing approaches to architecture description need to be enhanced as they fail to fully capture the unique characteristics of an enterprise system.

The architecture of a system is a specification that captures the structure and functionality of the system in an abstract manner. The specification should help various stakeholders to understand and analyze different functional and quality of service (QoS) aspects of the system before its construction, without providing overwhelming details.

In most software projects, the architecture specification of an enterprise system is captured using informal box and line diagrams with textual annotations. The informal architecture specification is transformed to implementation through a manual development process where the specification is hypothetically related to the implementation (through manual verification). Such a description suffers from people- and project-specific idiosyncrasies leading to ambiguity, misinterpretation and often, erroneous implementation. Therefore, it is essential to capture architecture specification using formal notations with well-defined syntax and semantics – through a tool.

A formal architecture specification should ideally capture all the aspects that are unique to the enterprise system and also help in reasoning various architecture decisions.

An enterprise system typically automates one or more business processes. The system consists of several legacy and new applications that are integrated to realize the consistent whole. The enterprise system is implemented using multiple technology platforms, commercial off-the-shelf (COTS) products that may be deployed in distributed hardware nodes.

There have been several attempts in the past to model different aspects of the architecture. For instance, there have been attempts to define what should be the essential constituents of the architecture[1]; the structural and behavioral

description languages for architecture specification[2]; notations to capture the information content[3]; transformation of the high level specification to implementation and so on.

An analysis of the popular modeling approaches reveals that the approaches do not offer a complete solution that models all the characteristics of an enterprise system in an elegant manner. This observation has motivated Infosys to define an integrated architecture modeling approach by bringing together the best practices of all the previous approaches.

The architectural characteristics of an enterprise system is analyzed and a modeling approach is introduced here to capture the architecture specification. The approach has been converted to the InFlux Architecture development methodology. A case study is presented to explain the salient features of the modeling approach .

ARCHITECTURE MODELING REQUIREMENTS OF ENTERPRISE SYSTEMS

To understand the modeling requirements, it is important to be aware of the software architecture description that needs to be captured. The Software Engineering Institute (SEI) observes that there is no universally accepted definition of software architecture, but there are several modern[5], and classical definitions[6] that are documented. Despite variations, these definitions suggest that the architecture of a system (not restricted to enterprise applications alone) must be able to capture the organization's functional elements, their interactions, the data elements used and transformed, and various non-functional or QoS attributes. It is widely recognized that architecture can be best described from multiple viewpoints[7] (Each viewpoint in

isolation addresses certain aspects of the architecture and helps in separating concerns, but when combined, form the system blueprint.

Key concepts

An enterprise system has multiple stakeholders, each interested in different aspects of the system. The business analyst, for example, is interested in the 'to-be' business processes and applications; the domain specialist is interested in identifying the functional applications and domain entities, and the operations department, which has to provision and deploy the infrastructure, may be interested in the deployment configuration of the hardware boxes and products (Table 1). The architecture modeling language should have a mechanism to capture these different aspects and the ability to represent each aspect in isolation. IEEE[8] further suggests that the architecture description should specify how the architecture meets "the concerns of the identified stakeholders of the system".

EXISTING APPROACHES AND THEIR LIMITATIONS

Many languages and notations have been defined for architecture modeling. It has been commonly understood, however, that the architecture, at a minimum, should be described as a set of interconnected coarse-grained components, rather than as its implementation details such as data structures and functions implementing algorithms[9].

Broadly, there are two categories of notations recommended for architecture description: object oriented (OO) notations and languages, and Architecture Description Languages (ADL). There are a few prominent industry standards that prescribe what architecture should model such as ISO RM-ODP 10746. The Zachman framework is another

Aspect	Description	Concerned Stakeholders
Business	Captures the business processes automated by the system to give it a holistic, process-centric view. This view should possess (but not be limited to) the process, the participants of the process: applications and people, the roles played by various users, the locations where the applications would be deployed, and so on.	Enterprise and LoB architects, Acquirer (customer), Business users
Functional	Describes various functional blocks (or subsystems) of the system, the functional services offered by them, their inter-relationships, the information exchanged by them, and so on. It should describe how overall functionality articulated by the acquirer of the system (typically client) has been distributed among these subsystems and how they interact among themselves to realize the functionality.	Architect, Domain Analyst
Information	Describes various information entities managed by the above mentioned functional blocks or subsystems and a high-level plan to realize these information entities.	Domain Analyst, Database Designer
Implementation on Technology & COTS products	Describes how the functional blocks and information entities should be implemented in terms of the platform and architecture styles (such as n-tier architecture), interaction protocols, the COTS products (middleware, functional products) , and so on.	Architect, Customer
Environment	Involves other existing systems (developed over time and deployed in distributed locations) with which the system needs to interact. The architecture modeling language should be able to model the environment and interaction with the environment.	Architect, Customer
Deployment	Describes the hardware (computing, data and network) infrastructure where the system would be deployed and managed. It describes how performance, scalability and reliability are achieved.	Architect, Deployment Manager
Development	Describes how the downstream developers will implement the system; principles and standards for adoption by the design teams and the release and configuration management guidelines.	Architect, Application Developer, Release & configuration manager
Design Rationale	In defining several architectural aspects especially the business, functional, information, implementation platform, technology and product selection, interaction with environment, deployment & development, the architect takes a series of architecture decisions. The architecture specification should formally record these architecture decisions so that they can be traced and analyzed at a later stage and help various stakeholders to understand the architecture better.	Architect, Application Developer, Customer
Quality of Service	Involves security, performance, portability captured in parallel with the functional specification. These aspects need to be considered and incorporated during architecture definition so that the development team can implement the QoS aspects during the implementation of the system.	Business users, Customer, Architect

Table 1: *Different aspects of enterprise system architecture from stakeholder perspective.*   **Source:** *Infosys Research*

standard that provides a set of views of the architecture.

Architecture Description Languages

Architecture Descriptive Languages (ADLs) such as C2, Wright, Darwin, Acme, and Rapide have been specially designed to describe the architecture of a system as an interconnection of coarse-grained components. Though most ADLs are based on formal semantic theory that helps verify and analyze the architecture models that are created using an ADL, there is little effort to standardize ADLs. ADLs lack the notations to capture all concepts of enterprise systems (Table 2). For example, ADLs do not capture the notion of architecture stakeholders and multiple views of a system.

Universal Modeling Language (UML)

The OO notation, especially UML, has become the lingua franca in the software engineering community to describe design elements in a modular fashion in terms of data types, algorithms, and procedures. The 4+1 view of architecture modeling, a precursor to UML, suggests that architecture should be described using four views. Objetct-oriented notations, similar to UML, have been used to describe these views.

While UML is very successful in modeling the implementation details, questions remain if UML is the right language for architecture description. It has been argued that UML in its current form does not provide adequate support to describe an architecture as interconnection of coarse-grained components where it is possible to model hierarchical decomposition, various communication aspects, architecture styles, non-functional properties and that it is possible to perform various architectural analyses. This observation has led to the effort of improving the UML specification to incorporate various architectural constructs in UML 2.0.

Other Methodologies

IEEE 1471 provides a definition of architecture for software-intensive systems and a conceptual framework to establish a normative set of architecture terminologies. It does not, however, prescribe any possible set of viewpoints and views, a modeling notation to capture them, or a process or method to describe the enterprise architecture.

ISO RM-ODP defines viewpoints for enterprise architecture. It does not identify the views and does not cover the aspect of downstream handoffs. While the Zachman framework provides a set of views of enterprise architecture, it is subjective, not formal and is not backed by notation.

The Meta-Model Approach

A meta-model-based approach to capture the architectural description can be used to overcome the above shortcomings. In this approach the architecture metadata is defined using the framework suggested in OMG MOF. The approach has been implemented by Infosys as the InFlux Architecture Development Methodology.

DESIGN PRINCIPLES

The architecture meta-model notations have been created based on the following set of design principles:

- **P1:** Adopt industry standards for normative set of terminologies and notations to define architecture. This helps create a standard set of architectural vocabularies for communication across various stakeholders.
- **P2:** Facilitate separation of concerns to

manage architectural complexities. Due to the diversity in architecture aspects it is a recommended practice to use multiple viewpoints and views to describe the architecture.

- **P3:** The metadata should contain modeling entities and notations close to an architect's notion of an enterprise system. The metadata should support evolution to introduce new concepts and notations.

- **P4:** Facilitate downstream integration. The notations should be unambiguous with well-defined semantics so that they can be incorporated in an architecture definition tool with the aim of downstream design tools integration.

- **P5:** Facilitate automation in various architecture analyses. The notations should have well-defined semantics so that it is possible to perform various analyses on the architecture model.

Based on these principles the architecture meta-model has been defined with the following characteristics:

1. **[P1,P2] Adoption of IEEE 1471:** The architecture definition has been adopted from IEEE 1471 standard. The conceptual framework provided by IEEE 1471 has been used as the basis for defining the modeling entities and the modeling tool.

2. **[P2] Adoption of ISO/IEC RM-ODP 10746 viewpoints for separation of concerns:** Since IEEE does not prescribe any viewpoint, it is left to the practitioner to select the appropriate viewpoints. The meta-model approach adopts ISO/IEC RM-ODP 10746 recommended viewpoints as these viewpoints closely match the enterprise system architecture aspects introduced in Table 1. Further, a new viewpoint called Software Organization Viewpoint has been introduced for better downstream design integration.

3. **[P3] Adoption of MOF for architecture metadata:** MOF suggests a four-layer approach to describe any kind of metadata. In the context of architecture modeling, the architecture metadata comprises of a set of architectural entities and their relationships defined using MOF meta-metadata.

4. **[P4] Facilitate Platform independent and platform specific modeling:** The modeling entities have been designed so as to clearly separate platform-independent and platform- specific aspects of the architecture with clear traceability relationships between them.

5. **Adoption of ADL concept:** The notion of components and connectors has been modeled using specific notations and these notations have been used in the engineering viewpoint. The component concept has been used for model application components such as EJB, .NET components, and architecture components such as middleware etc.

6. **QoS aspects with meta-model notations:** The proposed approach has adopted ISO 9126 recommended classification of various QoS aspects. Currently, the performance and availability aspects are under active consideration for QoS modeling in a formal manner.

7. **Design rationale captured with meta-model notations:** The meta-model enhances IEEE 1471 definition of design rationale. It introduces notations to capture

design rationale, architecture principles, architecture decisions, and so on. These notations are linked to different views and viewpoints.

8. **Architecture Traceability:** The meta-model entities associated with various viewpoints are also linked together so that these viewpoints can be integrated to form a consistent whole. The meta-model also associates various design rationale with the viewpoints so that it is possible to have reasons behind constructing a view in a particular manner.

IMPLEMENTATION OF KEY CONCEPTS

Viewpoints and views: The architecture of a system is defined using a set of diagrams that belong to RM-ODP viewpoints. Each viewpoint in turn is described using a set of views (or diagrams) and each diagram is defined using a set of notations (Table 2).

Use of Component and Connectors: The component and connector concepts introduced in ADLs are generic and abstract. ADLs do not prescribe which aspect of architecture should be modeled using these concepts since ADLs do not have concepts of viewpoints and views. However, the proposed approach uses component and connector notations in the engineering viewpoint.14

Architecture Principles and rationale: IEEE 1471 recommends design rationale for architectural concepts and consideration of alternative concepts but does not prescribe any approach for documenting the rationale. The proposed approach adopts the IEEE 1471 recommendation and provides a more prescriptive approach by defining additional design rationale notations (Table 3).

The design rationale entities are linked with each view (refer Table 2). By using this approach it is possible to associate the rationale precisely with the focused aspect of the architecture.

Architecture Traceability

Architecture traceability has been addressed at two levels:

- **Viewpoint traceability:** While a viewpoint-based architecture description facilitates the separation of concerns and helps manage the complexity, it is also necessary to have mechanisms to link all the viewpoints to define the consistent whole. A sample set of traceabilities are as follows:

  1. How can business processes be realized using subsystems?
  2. How are domain entity lifecycles controlled by subsystems?
  3. How can subsystems be implemented using technology?
  4. How can subsystems be deployed in the hardware infrastructure?
  5. How can subsystems be implemented as packages?

- **Design rationale traceability:** Entities in a specific architecture diagram have been identified based on certain design rationale. For instance, COTS products in an application diagram may be selected based on evaluation criteria. The clustering of devices in the deployment diagram may be necessary to meet certain QoS requirements. The domain analysis diagram may be influenced by a few architecture principles. The decision of breaking up a system into a set of subsystems can also be based on a certain design rationale. The meta-model facilitates capturing the traceability of design rationale (described in Table 3) behind diagram and diagram elements.

View point	Diagram	Description	Diagram Notations
Enterprise	Collaboration	High level view of the systems, the stakeholders of these systems and their interactions	Business process, activities (manual, automated, interactive) & tasks, workflow, roles, systems
	Work flow	Detailed business process involving the systems, and stakeholders (identified in collaboration diagram). The process is described in terms of activities performed by the system and stakeholders and work items exchanged.	
Computational	Context	Top level diagram to define system under consideration and its relationship with the environment	System, external system (derived from collaboration & workflow diagrams), subsystem, messages, service
	Functional	Hierarchical decomposition of the system into subsystems, their interaction and messages exchanged	
Information	Domain Analysis	Domain entities and their relationships at a high level (conceptual data modeling).	Domain entities and their relationship with the messages,
	Domain Lifecycle	Dynamic aspects of a Domain entity	location, UML relationships among the entities, state transition
Engineering	Logical	This diagram describes how subsystems and application components are organized based on an architecture style (such as n-tier, client/server, publish-subscribe), the architecture services required and so on.	Application and architecture components, services, connectors (simple and compound), required-provided interfaces, protocols, tier
Technology	Application	This diagram describes software platforms (operating system, middleware, database servers) and COTS products on which the application components and domain entities are deployed during runtime.	COTS product, operating environment, deployable unit (such as executable, shared library, deployable components like JAR, assembly)
	Deployment	Infrastructure required to implement the system such as hardware boxes, network links, database servers, firewall, routers etc	Computing, data and communication hardware, data center
Software Engineering (new viewpoint)	Layer	This diagram describes how the application components are further broken up into packages (or modules) and their dependencies	Application framework, package, layer, UML sequence diagrams
	Realization	This diagram illustrates how significant scenarios of system usage are executed using the architecture	

Table 2: Architecture modeling notations   ***Source:*** *Infosys Research*

QoS modeling

The proposed meta-model approach to architecture handles performance and availability as the QoS attributes. Other QoS attributes in ISO 9126 have not been modeled so far. The performance attribute of the model is further discussed below.

For performance modeling, each business

Notation	Description
Mission	The concept of a mission is adopted from IEEE1471 and enhanced with a specific set of attributes such as business driver, mission & plan, initiatives, and external constraints.
Architecture Principle	Architecture Principles are captured as general rules and guidelines to achieve the mission.
ArchitectureDecision, ArchitectureOptions, EvalCriteria	ArchitectureDecision models the decision taken while defining a View. A View can be associated with zero or more such decisions. A decision is made in two ways: 1. By selecting an option from one or more 'ArchitectureOptions' 2. Based on some evaluation criteria (EvalCriteria). The ArchitectureDecision notation is linked with the Diagram notations. Thus this notations helps one to describe the rationale behind the defining a diagram and the rationale behind selecting important architectural entities.

Table 3*: Modeling of Design Rationale*   **Source:** *Infosys Research*

activity is broken down to a set of atomic user tasks and system tasks. A system task is associated with an application service and certain performance attributes – expected mean / minimum / maximum percentile response times. User tasks are associated with think times. Each activity is associated with workload attributes like requests per hour for each operating time window based on business transaction volumes. This information can be used to execute capacity planning and infrastructure sizing algorithms and define a suitable infrastructure for the system.

ARCHITECTURE MODELING USING INFLUX

The architecture meta-data that was described previously has been realized in InFlux workbench, a modeling tool that is used to capture and analyze architecture descriptions. The enterprise viewpoint is captured as a collection of business processes in the workbench. The tool facilitates creation of other architecture diagrams from the business processes. The architecture toolkit feature classification framework suggested by Medvidovic and Taylor has been used to evaluate the features of InFlux workbench. The present capability of the tool with respect to the classification framework is as follows:

- **Information capturing:** The tool provides a visual means to define the architecture consisting of multiple views or diagrams (Table 3).
- **Active design:** The tool proactively supports an architect to manage consistency between the views. For instance, the tool helps in mapping an application component (identified in technology viewpoint) to an application server and eventually to the hardware.
- **Report generation:** The future version of the tool would generate the architecture document and various traceability analysis reports during the course of design. The traceability reports will be
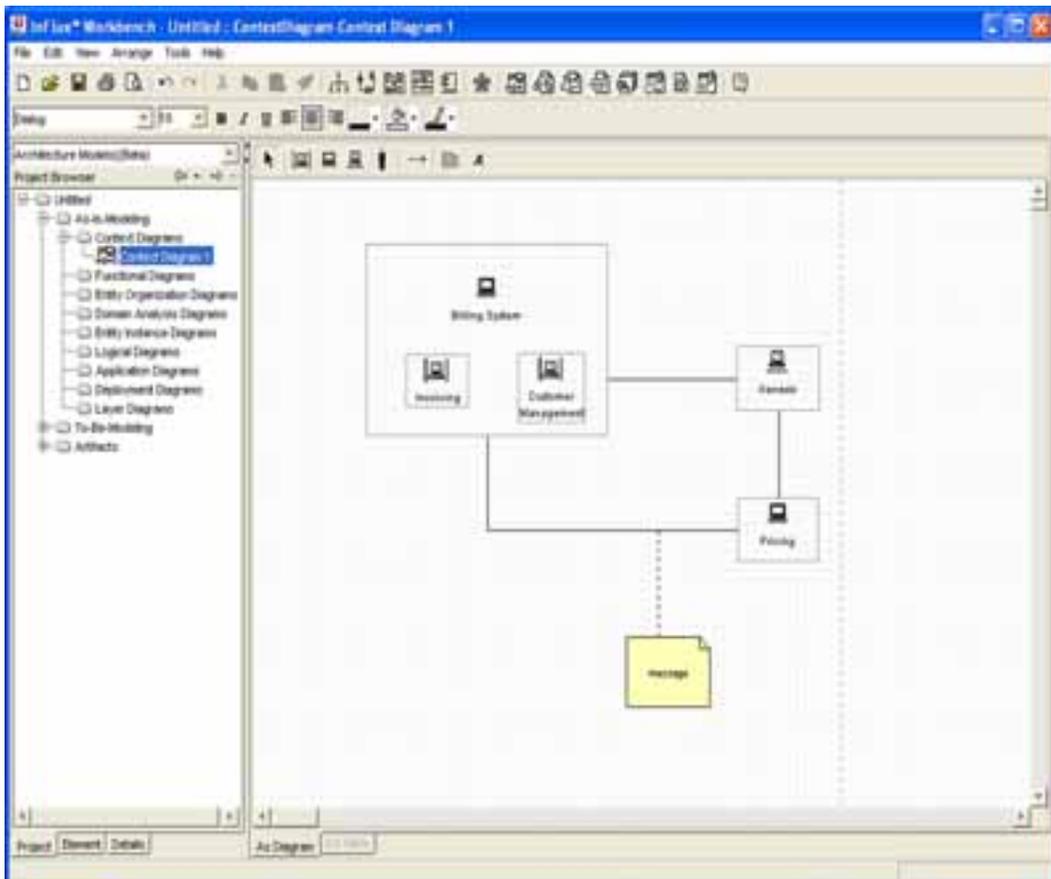
Figure 1: *Computational Viewpoint: A snap shot of the Functional Diagram of the billing system*

generated by analyzing various traceability aspects supported by the meta-model.

■ **Downstream integration:** The tool maintains the model information in XMI formats. The future version of the tool would be able to export the package structure for downstream development.

Modeling Illustration: A Case Study

An enterprise has initiated a global effort to create a single billing solution applicable to all regions. The new billing system will replace the existing collection of separate systems that currently support billing functionality by creating a single network-wide billing solution.

InFlux workbench helps the architect to construct different architecture diagrams (described in Table 2) as a part of 'Billing System To-Be' model.

Computational Viewpoint

The Computational viewpoint consisting of context and functional diagrams is based on the business processes (Figure 1). The systems shown in the functional diagram such as Billing System,
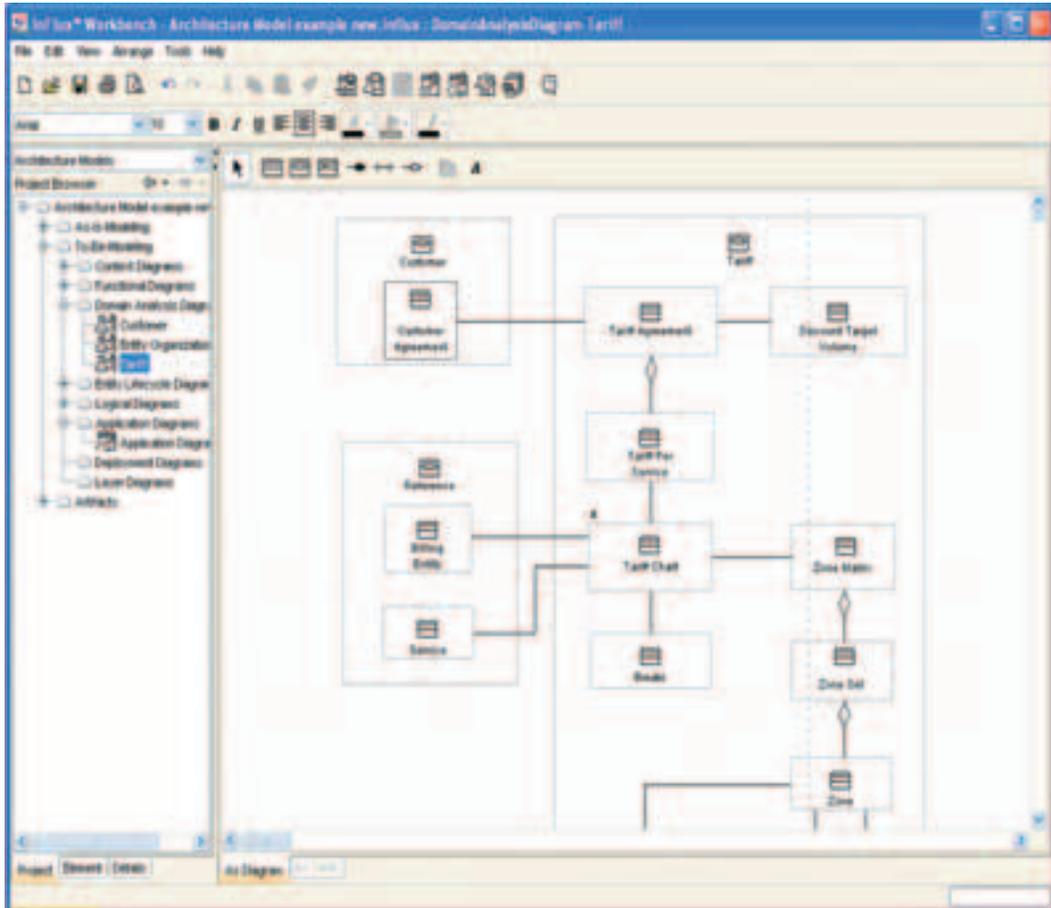
Figure 2: *Information Viewpoint: Sample Domain Analysis Diagram*   **Source:** *Infosys Research*

Genesis, and Pricing are identified in the business process. The subsystems are obtained by clustering the closely related activities defined in the business process.

Information Viewpoint

The Information Viewpoint consists of architecturally significant domain entities such as tariff, discounts, billing entity, and their relationships (Figure 2). The domain entities can be derived from the messages that are communicated among the subsystems identified in the Computational Viewpoint and the work products

identified in the workflow diagram (Enterprise Viewpoint). To capture the relationships, the Infosys' approach has adopted UML generalization, aggregation and association concepts.

Technology & Engineering Viewpoints

These viewpoints consist of logical diagrams, application diagrams and deployment diagrams. A logical diagram can be derived by mapping technology elements on the functional elements identified in the Functional Diagram (refer Figure 1). Here, the architect decides the architecture style (such as a three-tier solution), the architecture
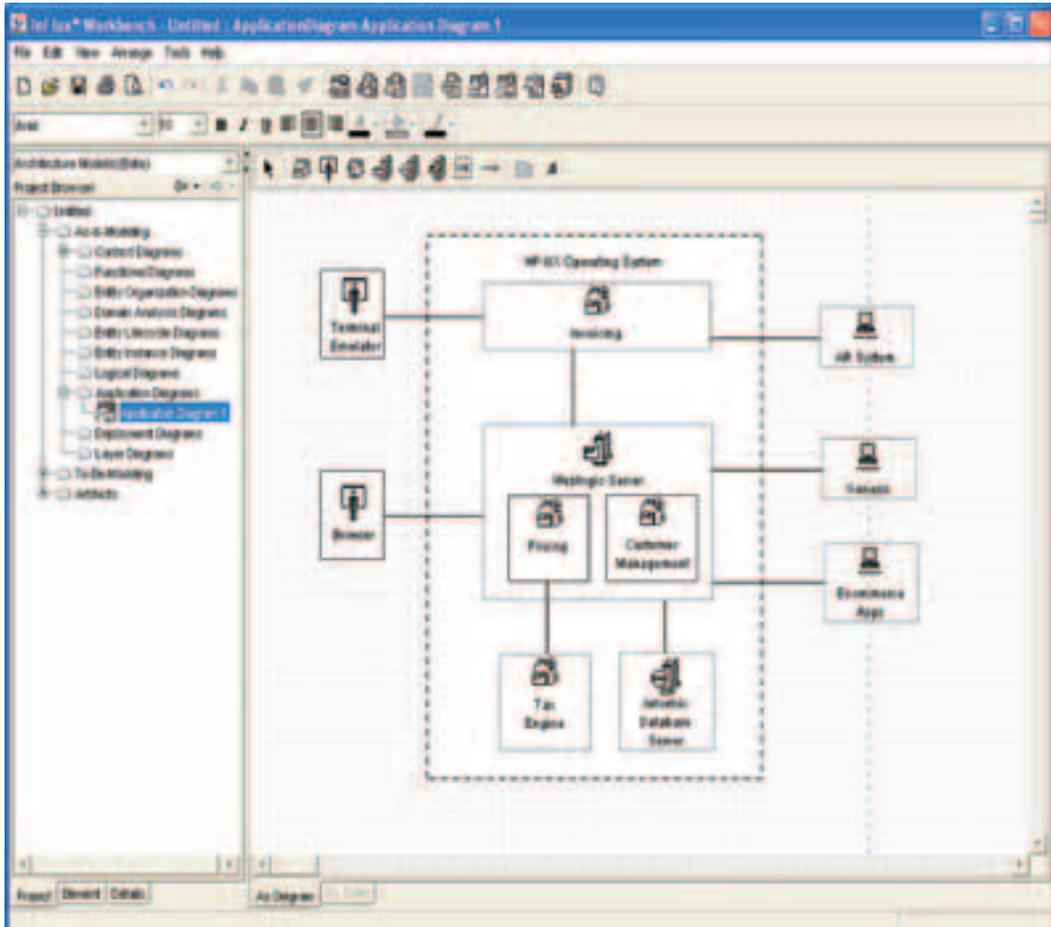
Figure 3: *Technology Viewpoint: Application Diagram*   **Source:** *Infosys Research*

services (such as presentation service, notification service, transaction service, and so on) that is required by the functional elements, communication aspects (API call, messaging, remote procedure calls and so on).

The application diagram provides a configuration of the deployable units that can be deployed either on hardware or software infrastructure. It shows all the processes for the system, the infrastructure software (middleware, application servers, and so on) that help run the application and the deployment units (JAR, DLLs, and so on). 'Pricing' and 'Customer Management'

applications are deployed on a J2EE Application Server. Invoicing is a separate executable and Tax Engine is a COTS product (Figure 3).

The Deployment Diagram depicts a view of how the processing nodes are arranged across locations and the software that will be installed on them. It depicts all the hardware and network connectivity requirements for the system and how the hardware is clustered and distributed across different geographical locations. The shows that the application is deployed in two separate Data Centers. Each Data Center will have a web server running on a machine and a cluster
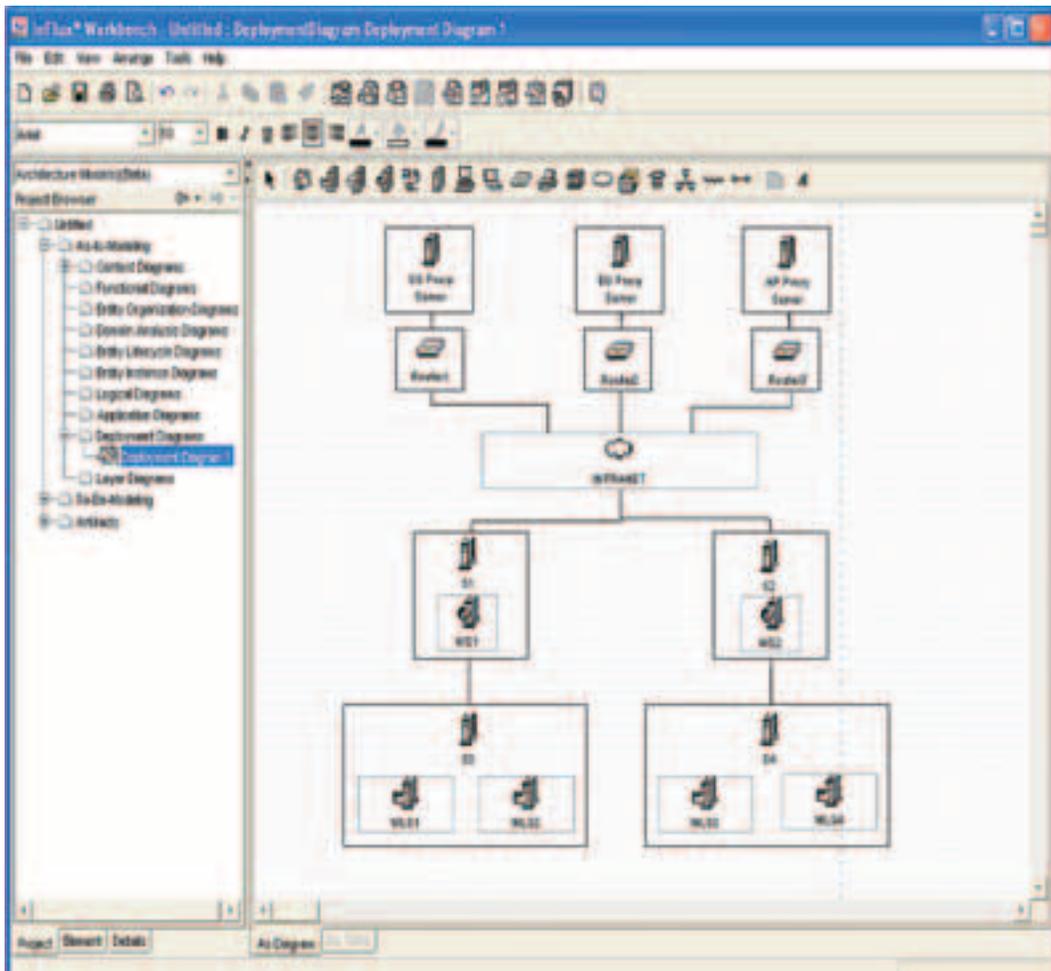
Figure 4: *Technology Viewpoint: Deployment Diagram*   **Source:** *Infosys Research*

of J2EE servers running on a separate machine (Figure 4).

COMPARISON TO OTHER APPROACHES

Existing architecture standards by themselves cannot address all aspects that are of interest to the stakeholders of enterprise systems. While ADLs, based on component and connector concepts, formalize the box and line diagrams, they do not clearly articulate how one can define business process, information, and technology aspects of the architecture and integrate these aspects to provide a consistent whole. Further, ADLs do not provide adequate support to model multiple views to describe different aspects of a system.

UML notations and tools on the other hand have been successful in downstream design. However, the modeling concepts provided by UML do not match the architect's vocabulary for system description. Besides they do not completely meet the needs of distributed component based architectures of today that involve platforms such as J2EE and .NET, Service-

Oriented Architecture (SOA) and integration middleware. This makes it necessary to integrate and extend these standards to meet architecture modeling requirements. The proposed approach is an attempt in this direction to provide a modeling language for enterprise systems that is based on distributed component architectures. The approach has integrated IEEE 1471, ISO RM-ODP, ISO 9126, UML, MOF and Component-connector concepts. It subscribes to the concept of views- and viewpoint-based architecture description suggested in IEEE 1471, RUP and 4+1 approach and RM-ODP. It has extended and tailored these standards for the needs of enterprise systems and supplemented these with graphical architecture notation and a modeling tool.

The fundamental concepts behind component-connector formalism has been used to describe the application components, their interface and interaction details. The UML aggregation and association relationships have been used to describe the information aspect of the architecture.

The proposed approach utilizes various architecture modeling concepts and enhances a number of architecture modeling aspects of other approaches while also introducing new aspects.

Business Architecture

The 4+1 approach of architecture modeling does not state how business architecture (involving business process and other associated concepts described in Table 1) can be modeled. EDOC profile defines notations, stereotyped from UML entities, to model business architecture as RM-ODP enterprise viewpoint. However, it does not prescribe how many views a practitioner should define for enterprise viewpoint. The popular ADLs do not adequately describe a mechanism

to capture the business architecture. The proposed approach suggests specific notations for business processes workflows to define enterprise viewpoint similar to EDOC. Moreover, it prescribes specific views as diagrams under enterprise viewpoint.

Functional Architecture

Most ADLs use component-connector concepts to model functional architecture. Many ADLs incorporate rigorous formalism in the language to perform various analyses such as functional simulation, verification and code generation. UML and UML-based architecture modeling approaches do not have such concepts. UML EDOC defines notations for component-connector concepts using stereotypes of UML class and association. The proposed approach uses notations similar to EDOC and introduces additional concept of system, subsystems, functional components and connectors (simple and compound) to model functional architecture. It treats component and connectors as first class modeling entities and these concepts are influenced by the ADLs. Unlike ADLs, the current version does not have the adequate formal rigor to capture the semantics.

Information Architecture

ADLs do not address information modeling adequately. The 4+1 approach does not contain a specific view to capture information aspects. It recommends ER diagrams and class diagrams for information modeling. EDOC provides notations for information modeling through RM-ODP recommended information viewpoint.

EDOC primarily uses UML class stereotypes for ER concepts, provides traceability of entities with functional elements (system, subsystems) and provides for grouping of entities and associating the groups with information

stores hosted in enterprise geographical locations for information modeling.

Implementation technology and COTS products

The proposed model supports explicit notations to model concepts such as architecture services, middleware, deployable units, communication aspects, and their traceability with the functional aspects. Most ADLs do not have adequate constructs to capture implementation technology, architecture services, and COTS products. The UML and 4+1 view approach suggests a process view to model deployable units and their communication but it does not

Deployment

This aspect of the approach supports an application diagram and deployment diagram in technology viewpoint to model deployment aspects. UML 4+1 approach has a Physical View that provides a subset of the deployment-related concepts. Other standards do not have an equivalent representation.

Development

An additional software organization viewpoint that is not supported by ADLs is also provided in the meta-model approach for better downstream design integration.

The meta model approach uses fundamental concepts behind component-connector formalism to describe the application components, their interface and interaction details

---

have equivalent notations for all the above mentioned concepts. EDOC focuses mostly on communication aspects (FCM model) and component distribution. It is not clear how an enterprise can model various other architectural aspects that are related to the implementation technology identified in Table 1.

Environment

The context and application diagrams are provided in the proposed approach to model the interaction between the system and its environment from functional and technology perspectives. There is no such equivalent concept in other standards.

Design rationale

The IEEE 1471 conceptual framework emphasizes the necessity of design rationale but the framework does not suggest any approach to capture it. The framework provides explicit notations to capture them. UML, EDOC profile and ADL do not have notations to capture such concepts.

Architecture traceability

The aspect provides for traceability among the views and viewpoints. The meta-model elements are connected through traceability relationships. UML 4+1 does not address traceability explicitly, but it is

possible to create such relationships. EDOC profile allows the designer to establish integration among different viewpoints. ADLs do not directly support the notion of architecture traceability. The extent of support is not clear.

QoS modeling

UML does not provide a mechanism to model QoS aspects of the architecture. Some ADLs capture QoS properties such as timing and security aspects. However the applicability is restricted to special class of applications such as real time systems. It is not clear if these concepts can be adopted in enterprise class applications. The proposed modeling language addresses some of the QoS aspects and currently it defines the performance and availability aspects.

CONCLUSION

The proposed modeling language in its current state does not have the complete capability to perform various architecture analyses. System generation from the architecture specification is only partial in terms of skeleton packages. Work is currently underway to enrich the framework. Rigorous behavioral semantics for various architecture analysis capabilities is required. It needs to support QoS modeling for the ISO 9126 attributes. It needs tighter linking with patterns. These will be the directions for future work in the architecture modeling framework.

REFERENCES

1. Booch, G., Jacobson, I., and Rambaugh, J., 1998. The Unified Modeling Language User Guide, Addison Wesley

2. Chen, P. P., 1980. Entity-Relationship Approach to Systems Analysis and Design, Proc. 1st International Conference on the Entity-Relationship Approach. North-Holland, ISBN 0-444-85487-8

3. Clements P., Bachmann F., Bass L. , Garlan D., Ivers J., Little R., Nord R., Stafford J., 2002. Documenting Software Architecture Views and Beyond, Addison-Wesley

4. Garlan, D. and Kornpanck, A. J., 2000. Reconciling the needs of architectural description with object-modeling notations, in Proceedings of the Third International Conference on the Unified Modeling Language – 2000

5. Hillard, R., 2001. IEEE Std 1471 and Beyond, Position Paper SEI First Architecture Representation Workshop

6. IEEE1471, 2000. IEEE Standard 1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, September 2000

7. InFlux Architecture Development Methodology (http://setlabs/ ArchitectureModeling/)

8. ISO10746-2, 1996. ISO/IEC 10746-2: 1996, Information technology.Open distributed processing.Reference model:Foundations.

9. Kandé, M. M., Crettaz, V. A., Strohmeier, and Sendall, S., 2001. Bridging the gap between IEEE 1471, Architecture Description Languages and UML, Springer Software and Systems Modeling Journal, Vol 1 Issue 2 pp 113-129 Special Issue UML 2001

10. Kruchten, P., 1995. Architectural Blueprints-The 4+1 View Model of Software Architecture, IEEE Software, vol 12, no 6, pp. 42-50

11. MDA, 2001. Model Driven Architecture- A Technical Perspective, OMG Architecture Board MDA Drafting Team, Review Draft Version 00-17

12. Medvidovic, N., and Taylor, R. N., 2000. A Classification and Comparison Framework for Software Architecture Description Languages, IEEE Trans. Software Engineering, vol. 26, pp. 70-93

13. MOF, 2002. MOF Conceptual Overview, OMG Meta Object Facility v 1.4.

14. Perry, D. E., and Wolf, A.L., 1992. Foundation for the Study of Software Architectures, SIGSOFT Software Eng. Notes, vol 17, no 4, pp. 40-52

15. Sarkar, S., and Thonse, S., 2004. -EAML Architecture Modeling Language for Enterprise Application, IEEE Conference on E-Commerce Technology for Dynamic E-Business

16. SEI, 2003. How do you define software architecture, SEI on-line publications, available at: http://www.sei.cmu.edu/architecture/definitions.html

17. Shaw, M. and Garlan, D., 1996. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall

18. Stojanovic Z., Dahanayake A., and Sol H., 2001. "Integration of component-based development concepts and RM-ODP viewpoints", in 1st workshop on Open Distributed Processing 2001, Setbul Portugal, pp 98-109

19. Störrle, H., 2001. Towards Architecture Modeling with UML Subsystems, Ludwig-Maximilians-Universität München

20. Stutz, C., Siedersleben, J., Kretschmer, D., and Krug, W., 2002. Analysis beyond UML, in IEEE Joint International Conference on Requirements Engineering (RE'02), September 09 – 13, Essen, Germany, pp. 215 – 218

21. UML2.0., 2000. UML 2.0 Superstructure RFP, OMG Request for Proposal, OMG document ad/00-09-02

22. Zachman, J. A., 1987. A Framework for Information Systems Architecture. IBM Systems Journal, vol. 26, no. 3. IBM Publication G321-5298

Authors featured in this issue

SANTONU SARKAR

Santonu Sarkar, PhD, is a Senior Technical Architect at SETLabs. His research interests include architecture modeling and analysis, agent technology, formal modeling, QoS analysis, and web services. He can be reached at santonu_sarkar@infosys.com

SRINIVAS THONSE

Srinivas Thonse is a Principal Archìtect at SETLabs. He engages in research projects in the areas of IT methodologies, software architecture and business process management. He can be reached at srinit@infosys.com

Infosys®

POWERED BY INTELLECT

DRIVEN BY VALUES