Testable Integration Architecture: The Power of Abstraction

Executive Summary

In this paper, we describe Testable Integration Architecture (TiA), a unique capability supported by an open source toolset and an accompanying methodology. Specifically, TiA can be applied to the critical area of communication models for software systems -- we might call this the integration architecture -- and it enables decision makers to address issues before they become defects. TiA empowers one to think in global terms, focusing on system behavior by taking a systemic view. TiA can provide insights into the consistency of requirements, as well as the surety of process designs and their attendant integration concerns, through the testing of models against requirements. The benefits to customers include early defect detection and consistency of requirements, achieved through model testing. This results in higher alignment, lower costs, faster delivery, higher quality and a clearer return on investment.

The traditional methods of addressing quality and decreasing time to market require higher levels of investment in governance and skills. TiA does not require any additional investment, and this is what makes it fundamentally different because it reduces overall costs, while also reducing time to market and increasing quality.

In a service-oriented architecture (SOA) context, TiA is a concrete means of ensuring that services meet their business requirements, a key principle mandated by the "SOA Manifesto." SOAs are defined by the presence of service contracts, which describe what a service provides in terms of functions, inputs and outputs. The role of TiA is in the composition of service contracts as a set of peered services that compose a solution. In this regard, the TiA description provides an integration model that maps out the communication pertaining to service contracts that can be tested against business requirements. By doing so, it ensures the best use of existing service contracts and the best design for new service contracts that make up a solution.

Overview

"The best SOA innovation I have ever seen." – Chief architect, global insurance company

Testable Integration Architecture combines the power of abstraction and formal methods to allow integration architectures to be better described. It enables them to be formally and rigorously tested to ensure that they meet business requirements.

TiA describes the fundamental interactions and ordering rules that need to occur between a set of application(s) or services to deliver a solution.

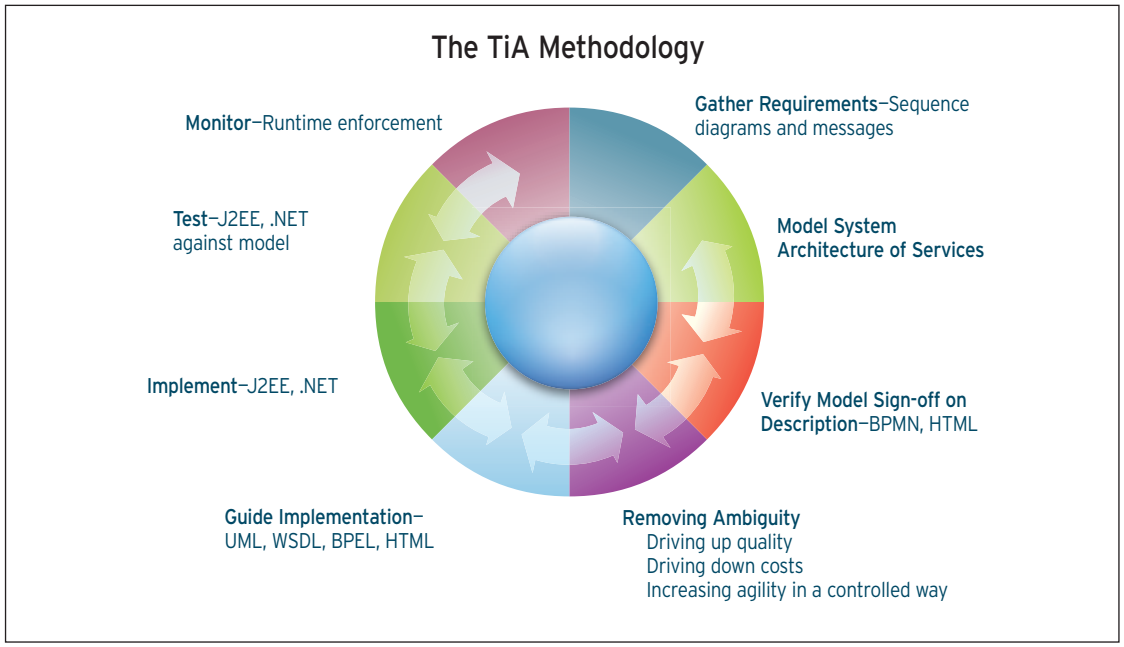


Figure 1

Using TiA can reduce the risk of delivering solutions that don't match requirements, inadequate testing, high cost of delivery and ongoing maintenance in the face of change. A Testable Integration Architecture can also increase the quality and flexibility of a solution.

These benefits are achieved through a formal alignment of the integration architecture with the requirements, through testing, which reduces the risk of design time errors.

Furthermore, an alignment of the solution with the integration architecture results in runtime alignment, which enables stronger management of change over time.

While the tool suite is open source, the methodology for Testable Integration Architecture was developed by us over the last two years. The leadership that we continue to provide in framing the development of the tools and refinements to the methodology is a testament to the visionary approach we have applied to the field of enterprise architecture as a whole.

About Testable Integration Architecture

Testable Integration Architecture is supported by an open source tool suite -- the collaboration tool suite from Redhat (www.jboss.org/savara) -- and the methodology we developed.

Here are the questions that Testable Integration Architecture addresses:

- Are our requirements consistent and implementable?
- Will my SOA platform support my target solution, and if not, why not?
- How many requirements does my integration architecture meet?
- What improvements can I make to my integration architecture to support my business?

Here are the benefits of using Testable Integration Architecture:

- Reduced cycle time for a solution.
  - By leveraging the power of abstraction in both requirements gathering and creating solutions with TiA.
- Provable solution against requirements.
  - Through formal testing of the integration architecture against requirements.
- Reduced implementation time.
  - Through the generation of technical contracts from the integration architecture that drive implementation.
- Reduced testing time.
  - Through the guarantee that technical contracts are complementary against the integration architecture.

- Through the testing of components against the integration architecture.

- Continual real-time governance with change management.
  - Through the monitoring of implemented solutions against the integration architecture over change.
- Reduction of risks associated with delivery.
  - Through the generation of technical contracts tested against requirements.

Engagement Model

We provide a range of engagements for clients of Testable Integration Architecture, including:

- Introductory workshops that last two to four hours.
- Targeted workshops that incorporate customer problems and span two half days.
- Sub-line-of-business solutioning that typically runs six to 10 weeks.
- Line-of-business solutioning that typically runs 12 to 20 weeks
- Cross-line-of-business solutioning that typically lasts more than one year.

The aim of our engagement pyramid is to ensure that customers understand the tools, methodology and benefits of our approach and gain a better appreciation of our global delivery model in bite-sized chunks.

The Methodology

Our methodology, along with the open source tools that form Testable Integration Architecture, provide successive refinements of both requirements and integration architecture models of solutions. This turns the process of requirements gathering to artifacts that drive delivery and implementation into a repeatable and manageable process.

The successive refinement mechanism is applied to requirements and integration architecture models of those requirements. The requirements gathered at each level are used to drive the development of an integration architecture model that expresses the solution at the requirements level. The integration architecture model is testable and/or verifiable against those requirements. At the higher levels, the verification is syntactic simply because there are not enough semantics. But as the levels go deeper, verification becomes semantic and is based on formal testing of those semantics as they are expressed in the requirements and manifested in the integration architecture model.

Formal testing of an integration architecture model against requirements is achieved by the simulation of the requirements against the integration architecture model. When deviations occur between the integration architecture model and requirements, they are highlighted in the requirement. The marking of deviations more easily facilitates discussion of the requirements with business analysts and subject matter experts and more easily facilitates design errors for the solutions architect. In some cases, the error is clearly a design error, and the solution architect can fix the integration architecture model and re-test. In others, it may be a misunderstanding of a requirement that gives rise to the deviation or, indeed, that the requirement is circumspect and the solution architect is able to rapidly engage with business analysts and subject matter experts to ensure corrective action is taken and then re-tested.

The levels of refinement are numbered 0 to 5. Here is a description of each level.

- Level 0 is the "business" level and frames a business context.
- Level 1 is the "lifecycle" level and frames the order in which high-level, granular business processes or activities should occur.
- Level 2 is the "landscape" level, which extends Level 1 by adding finer grained details to the business processes and activities in terms of the existing applications that play a role in a specific business process or activity. At this level, we add the idea of message flow, a key component of integration architectures, as opposed to focusing exclusively on business process and activity ordering. We call this level an "abstract" model of the solution because it has everything except the bindings to target platforms and the bindings to the underlying information model that drives the message flows.
- Level 3 is the "technical" level. It extends Level 2 by adding the bindings to the underlying target platform in the form of the technology standards to which they must adhere (e.g.,

WSDL1.1, WSDL2.0, BPEL1.0, BPEL2.0, etc.), as well as the underlying information model (e.g., XMLSchema, CVS, etc.) to which the messages that are exchanged in the flow must conform, in terms of the formats and -- very importantly -- the business transaction identities or correlation sets of identities over the flow. We call this a "concrete" model because it is grounded upon a platform and an information model. It is at this level that we are able to simulate requirements, in the form of sequence diagrams with example messages, over the integration architecture model to show that the integration architecture model does or does not support the requirements. This is what we mean by Testable Integration Architecture, in as much as the integration architecture model can be formally shown to fully support the requirements.

For the purpose of simplicity, we use the term "model" rather than "integration architecture model" to make it easier to read and understand the refinement process.

Defining the "Business" Level

At the highest levels, which we call the "business levels," requirements are a set of statements. That set is known as "R0." A model of the R0 requirements is known as "L0." The intention here is simply to capture the R0 requirements in some diagrammatic and annotated form. Generally, R0 requirements derive from high-level statements that are created by an executive or management board. An L0 model provides some pictorial representation of what might need to change. This would indicate the project scope, which drives budgetary and resource decision making.

A typical set of R0 requirements is illustrated below (see Figure 2), along with the L0 model that embodies them. L0 models are a convenience rather than a necessity for delivering a solution, and with no formal semantics behind them, they play no role other than indicating the scope of change.

R1 and L1: The "Lifecycle" Level

The first refinement step that we make is to gather what we call "lifecycle" requirements, known as "R1" requirements. The R1 requirements must be grounded upon the R0 requirements. In the example above, requirements pertaining to new policies would be outside the allowable refinement from R0 to R1, whereas refinements based on claims would be allowable. "Lifecycle" requirements are statements that define an ordering of key steps or business processes needed to deliver the R0 requirements. An L1 model, which is the refinement of L0 and is based on the R1 requirements, embodies the R1 requirements.

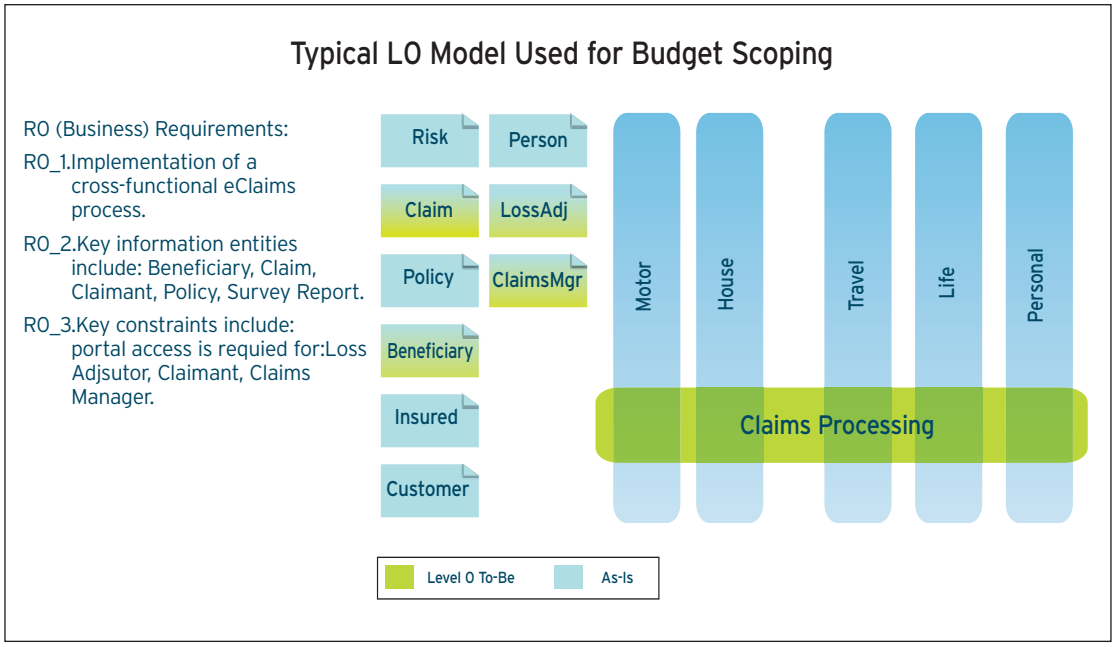


Figure 2

L1 Lifecycle Model Depicting the Lifecycle of a Claim

R1 (Lifecycle) Requirements:

R1_1:A claim must have been made before any further processing of a claim can proceed.

R1_2.Once a claim has been made and validated (see FirstNotificationOfLossFree.scn), the claim may be subject to any number of status enquiries by any authorized entity (Loss Adjuster, Claimant, Claims Manager).

R1_3.Once a claim has been made and validated (see FirstNotificationOfLossFree.scn) the claim may be processed by an authorized entity (Loss Adjuster, Claimant, Claims Manager) where processing advances the status of the claim.

R1_4.Status enquiries and claims processing may happen concurrently.

R1_5.A claim is deemed to have terminated once it has reached a conclusion.

R1_6.A conclusion may result in a payment to one or more parties.

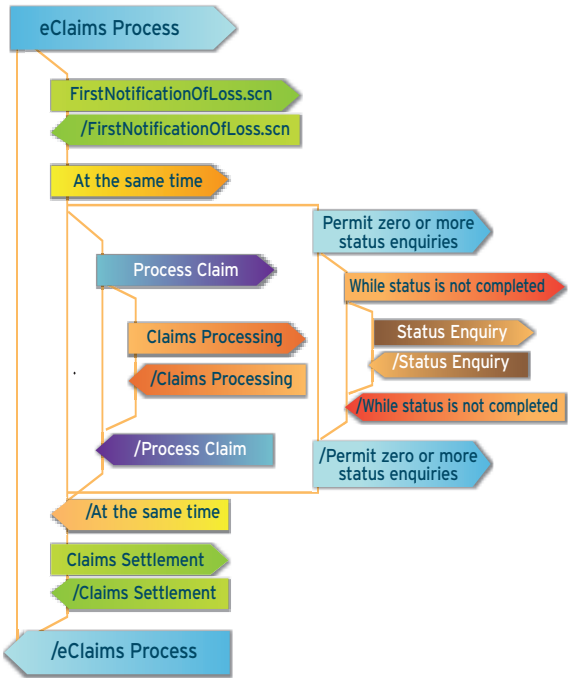


Figure 3

The R1 requirements and L1 model must be verifiably aligned; that is, the L1 model must express all lifecycle ordering constraints expressed in the R1 requirements. Although not formally testable at this stage, the review of R1 requirements against an L1 model must be able to show alignment by inspection.

An example of R1 requirements and the L1 model are shown above (see Figure 3). The L1 model has clear semantics in terms of ordering and successive refinements of both requirements. Both models will preserve the semantics expressed at the higher levels as we descend through the lower levels.

Now that we have a more formal model for our L1, we can start to do things with it. In our Testable Integration Architecture approach, we can generate reviewable artifacts from the L1 model and review against them. We are guaranteed that anything we generate is functionally equivalent to the L1 model, and so we might think of any generated artifacts as being a subset or subtype of our L1 model. We show a simple HTML example of the L1 model that was generated from that model below (see Figure 4):

Collaboration: eClaims Process

Choreography flow for the eClaims process

This collaboration is the "roof" collaboration, which means that it is the first one that is activated when monitoring or executing the business protocol defined within the collection of collaboration modules in the CDL package.

Activities

- Perform the first notification of loss collaboration by claimant.
- At the same time:
  - Process claim
    - Perform the claim processing collaboration
  - Permit zero or more status enquiries
    - Allow a status enquiry (condition is non-observable based on some status in the workflow for the claim)
    - Perform the status enquiry collaboration
- Perform the claim settlement collaboration

Figure 4

R2 and L2: The "Landscape" Level

The process of refinement continues to what we call R2 requirements and L2 models. The R2 requirements are known as "landscape" requirements. They typically constrain the solution of an existing landscape in which some components of the solution are already available. The existing components may be applications with APIs or services with WSDL descriptions. In the former, the aim of the solution is to service-enable the existing application, and in the latter, to re-use the service as-is.

In the R1 requirements, we had no notion of components, services, roles or actors. We simply stated what the key steps should be and provided some constraints on the ordering of those steps. In the R2 "landscape" requirements, we are more precise. In the case of First Notification of Loss, we give examples of which applications/services should be used and what the flow of information should be between them. We do this by drawing sequence diagrams (sometimes known as message sequence charts), which describe the order in which applications/services exchange data to implement a process such as First Notification of Loss.

Typically, this level of requirement is initially framed by a list of relevant applications and services that must be reused, as seen below:

```
R2 (Landscape) Requirements:
   R2_1:Legacy Policy System, Workflow System,
      Accounting System and Claims System.
```

The sequence diagrams that we require are not just pictures of swim lanes and message links between them. Example messages are also needed as an additional item per message link. Thus, each link has an example message. At this stage, the content of the messages is not important – they could be blank – but they will become important as we further refine towards a solution.

Typically, the set of sequence diagrams is not enough to be sure of a design; rather, they are indicative. For example, the key flows may be described, leaving many erroneous (unhappy) paths unstated, and yet the solution and the L2 model must ensure that these are also covered. Two example sequence diagrams are shown below (see Figures 5 and 6), which show a happy and unhappy path.

The process by which the L2 model is created from the R2 requirements has a strong degree

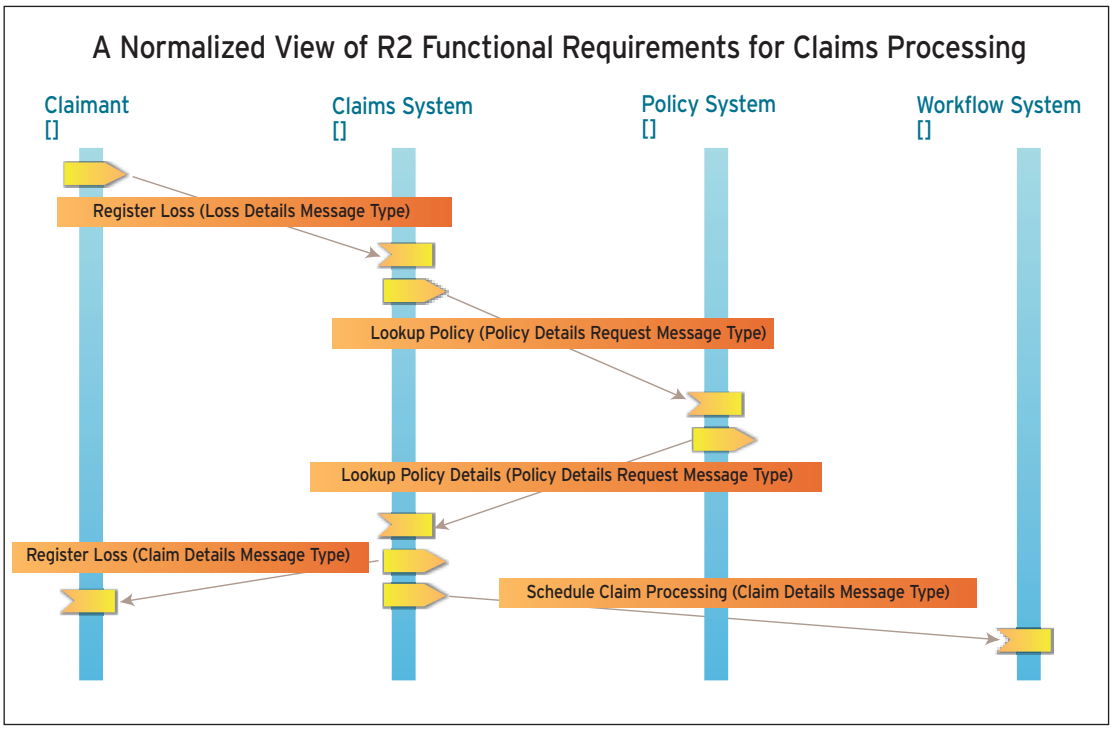


Figure 5

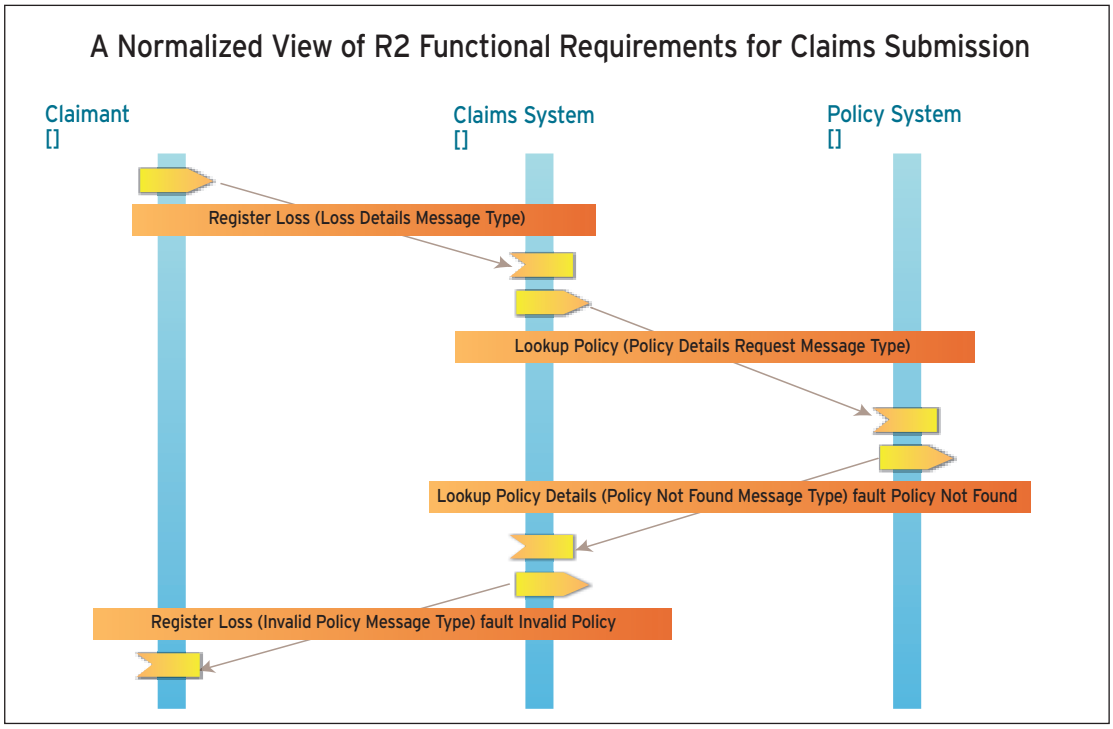


Figure 6

of collaboration between the subject matter experts, the business analysts and the architects. The process actively encourages collaboration to ensure that those things that are unstated are teased out and correctly modeled. In this way, the dialogue between the different parties to a solution is correctly framed and focused. This, in turn, leads to better alignment and removes ambiguity.

The starting point for the L2 model is -- for every swim lane across all sequence diagrams -- the necessary roles that are needed. In addition, for each pair of swim lanes, the starting point is also an exchange to show a relationship. This is done graphically and results in a static picture of the key components that are necessary to support the landscape as it has been defined, as well as additional requirements. In the insurance industry example (see Figures 7 and 8), these are portal access for the claimant, the claims manager, the loss adjuster and so on.

The static model embellishes the L1 model of processes and their ordering rules. Further embellishment is made to add the detailed flows as described by the sequence diagrams on page 8.

R3 and L3: The "Technical" Level

The refinement continues to R3, or the "technical" requirements, in which technical considerations are added to the requirements, such as business transaction identities over the set of applications/services and the target platform semantics (e.g., event driven or request response, WSDL, Java, BPEL, etc.).

The embodiment of the R3 requirements is the L3 model, which appears remarkably similar to the L2 model. The key differences are that the properties on the model may be directed to the target platform semantics, and the model is bound to a concrete information model.

In the first instance, the target platform semantics and errors in design relative to the target platform are found through model-checking against those semantics. A classic case is modeling intervening activities between a request and a response and wanting to use BPEL orchestration as a target platform. The two cannot be done because BPEL does not allow it. Another might be to use WSDL1.1, but the solution requires notification, which is not supported in WSDL1.1.

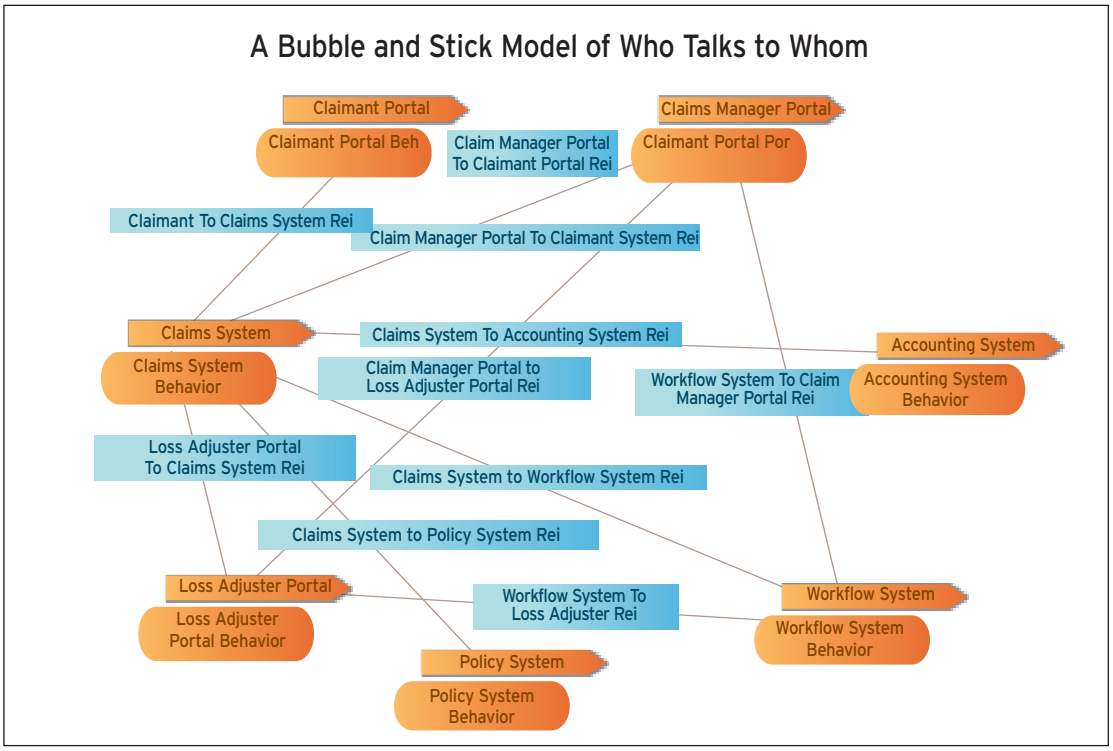


Figure 7

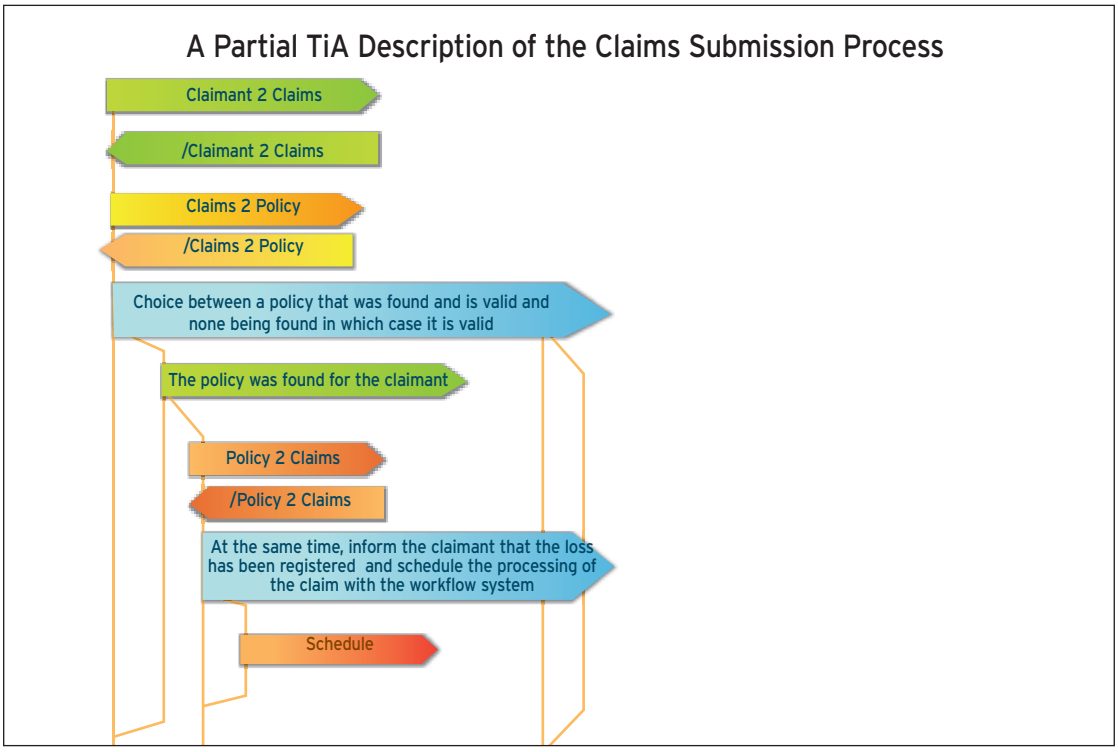


Figure 8

Collaboration: eClaimsProcess.First NotificationOfLoss

Collaboration for the first indication of a loss by the claimant

Activities

- Notification Of Loss Sequence
  - Register the loss with the claims system
  - Claims system validates the policy details for the claimant based on the loss details by checking in the policy system
  - Choice between a policy that was found and is valid and none being found in which case it is invalid
    - The policy was found for the claimant
      - The policy system returns a matching policy to the claims system
      - At the same time, inform the claimant that the loss has been registered and schedule the processing of the claim with the workflow system
        - Schedule
          - The claims system schedules the claim with workflow system
        - Inform
          - The claims system informs the claimant that the loss has been registerd and issues a suitable claim reference for the claims and matching policy
    - The policy was not found for the claimant
      - The policy system returns POLICYNOTFOUND to the claims system, indicating an invalid policy from the claimant
      - The claims system informs the claimant that the policy details are INVALID

Figure 9

In the second instance, the binding to an information model -- which must be done -- is achieved by explicitly stating for every message type what the underlying business transaction identities should be. This latter stage, the binding of identities, is more profound than at first it might seem. For example, in a claims process transaction, the first notification of a claim by the claimant to the claims system has some nominal identity based on the claimant's policy (e.g., PolicyId). The policy details provided are used to verify that the claimant can claim against a valid policy. If the policy is valid, the identity from that point on is tracked by a claim reference (e.g., ClaimId), and the claimant is asked for this detail in every subsequent communication with the claims process as a whole. So, what we see are changing identities over time and based on context.

Many systems that get this wrong suffer from problems when they are subject to integration testing. Sadly, the cost of remediation at that juncture can be high. By testing at this point, we ensure that the design that is embodied by the L3 model is correct and that the requirements in terms of message order and correlation identity sets are also correct.

Above is an example binding of a specific message type (see Figure 9). In this example, the Xpath expression simply states that the identity for messages of this type are to be found at the attribute known as ClaimId in the message.

Reviewing Solutions

At this point, we have a tested model, L3, which is available for review. Rather than reviewing the diagrammatic notation, which to the uninitiated may be intimidating, we enable the models at the various levels to be used as input to the generation of various forms of HTML textual documentation that describes the processes and flows and their ordering rules. This is done in ways that are more understandable for business-focused and technology-focused stakeholders. An example (see Figure 9, above) illustrates the extracts of the L3 business views.

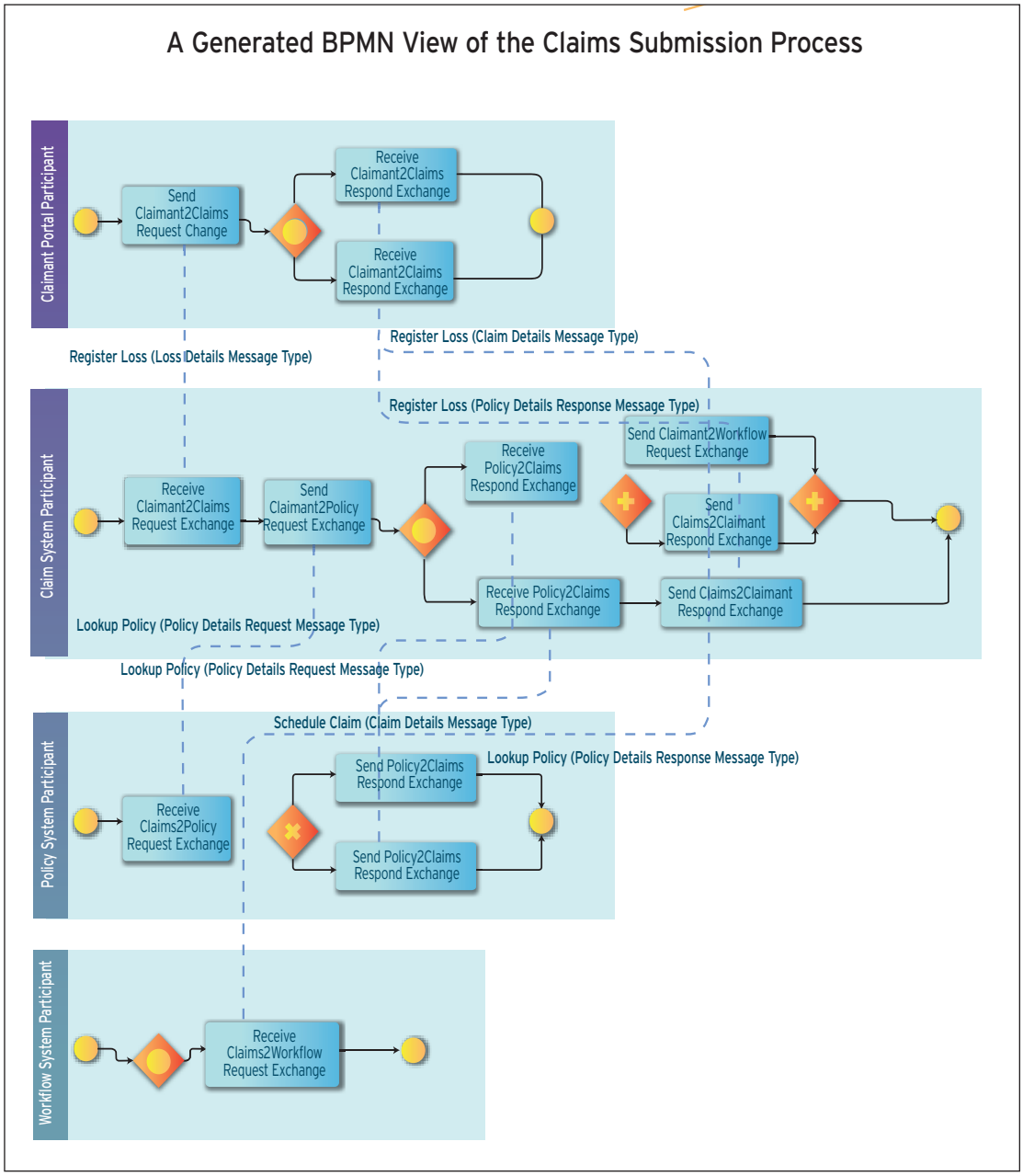


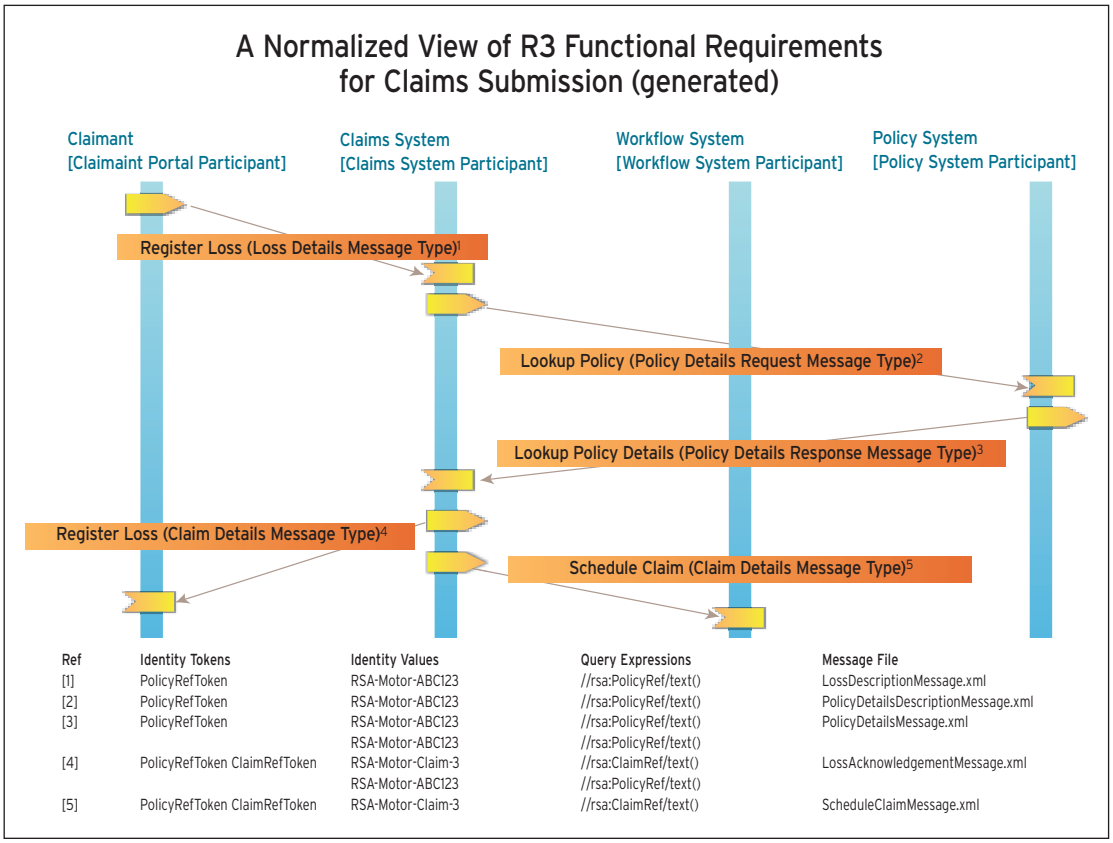
Figure 10

Equally, other formats can be generated. Here we show a BPMN fragment for the same process (see Figure 10).

Technical Contracts to Drive Implementation

Once the L3 model has been tested and reviewed, we can generate the technical contracts from the L3 model, safe in the knowledge that the artifacts accurately reflect the model and have the same validity -- based on formal testing and review -- as the model from which they are generated.

The aim of the technical contracts is to hold invariant the collaborative behavior of the different components that delivery teams will implement. These contracts do not define the internal business logic that they need to design and/or write; rather, they ensure that the external interfaces to other components, embodied by the integration architecture, are held invariant, which increases the chances of successful integration of the components and reduces failure of interoperability and collaborative errors. As a result, the number of passes through systems integration testing declines.



Ref	Identity Tokens	Identity Values	Query Expressions	Message File
[1]	PolicyRefToken	RSA-Motor-ABC123	//rsa:PolicyRef/text()	LossDescriptionMessage.xml
[2]	PolicyRefToken	RSA-Motor-ABC123	//rsa:PolicyRef/text()	PolicyDetailsDescriptionMessage.xml
[3]	PolicyRefToken	RSA-Motor-ABC123	//rsa:PolicyRef/text()	PolicyDetailsMessage.xml
		RSA-Motor-ABC123	//rsa:PolicyRef/text()	
[4]	PolicyRefToken ClaimRefToken	RSA-Motor-Claim-3	//rsa:ClaimRef/text()	LossAcknowledgementMessage.xml
		RSA-Motor-ABC123	//rsa:PolicyRef/text()	
[5]	PolicyRefToken ClaimRefToken	RSA-Motor-Claim-3	//rsa:ClaimRef/text()	ScheduleClaimMessage.xml

Figure 11

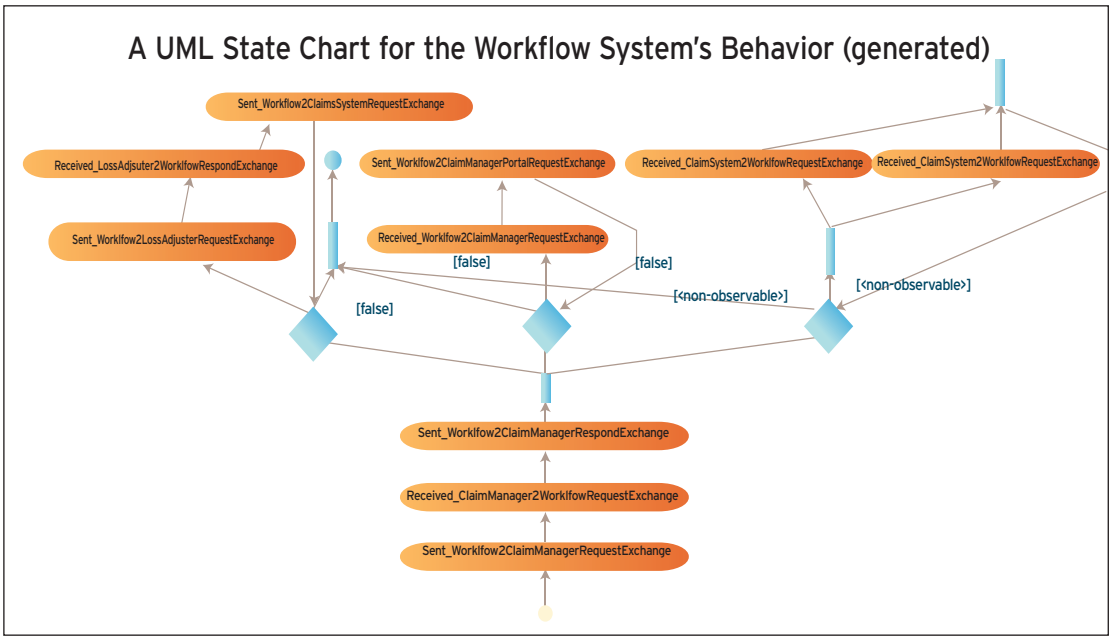


Figure 12

WSDL for the eClaims Workflow Participant

```
<?xml version="1.0" encoding="UTF-8"?
<definitons xmlns="http:schemas.xmlsoap.org/wsdl/"
     smlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns-
     "http://www.cognizant.com"
     <message name="">
         <documentation>
             pi4soa: user defined definitions
         <documentation>
     </message>
     <portType name="WorkflowSystemBehaviotPort">
         <operation name="NotifyReserve">
            <input message="tns:null" name="ClaimManager2WorkflowRequestExchange"/>
            <output message="tns:nell" name="Workflow2ClaimManagerRespondExchange"/>
         </operation>
         <operation name="UpdateClaimsStatus">
            <input message="tns:null" name="ClaimManager2WorkflowRequestExchange"/>
         </operation>
         <operation name="NotifyUpdateReserve">
            <input message="tns:null" name="Claims2WorkflowRequestExchange"/>
         </operation>
         <operation name="StatusRequest">
            <input message="tns:null" name="Claim2WorkflowRequestExchange"/>
            <output message="tns:null" name="Workflow2ClaimManagerRespondExchange"/>
            <fault message="tns:null" name="ClaimRefNotFound"/>
         </operation>
     </portType>
```

Figure 13

Examples of WSDL and UML State Charts, as well as grounded sequence diagrams, are shown in Figures 11, 12 and 13. All are generated from either the model or the business requirements.

Conclusion

This paper demonstrates, in a scientific and concrete manner, the application of TiA in managing the problem of system integration into distinct levels of abstraction. The power of abstraction is not a trivial task to instigate. On the one hand, it fundamentally governs the need for correctly representing the requirements of applications. On the other, it requires the foresight to design flexible structures in areas where requirements might change in the near future. TiA provides a global view of the system dynamics, offering the foresight needed to design solutions that will evolve as the landscape changes.

As a case study, we demonstrated how TiA has been employed in the insurance domain, focusing on the problem of claims, to moving from several distinct models of claims to an enterprise model, which overarches several lines of business. TiA provided a global view -- a systemic view -- of the problem across several lines of business, abridging the complex task of abstraction.

About the Authors

Steve Ross-Talbot is the European Technology Officer with the Global Technology Office for Cognizant. He is also an author of the SOA Manifesto, a pioneer in distributed computing, the founding father of WS-CDL and Testable Integration Architecture and an invited expert on many vertical standards, from ISO to HL7. Steve can be reached at Steve.Ross-Talbot@cognizant.com.

Dr. Bippin Makoond is a Senior Solution Architect at Cognizant who functions as Global Innovation Lead within the company's Banking & Financial Services practice. He holds three patents within the domain of wireless distributed systems and messaging technologies and is a visiting scholar and expert advisor to the Component and Distributed System Research Group (CODIS) at Kingston University. Bippin can be reached at Bippin.Makoond@cognizant.com.

About Cognizant

Cognizant (NASDAQ: CTSH) is a leading provider of information technology, consulting and business process outsourcing services. Cognizant's single-minded passion is to dedicate our global technology and innovation know-how, our industry expertise and worldwide resources to working together with clients to make their businesses stronger. With over 50 global delivery centers and more than 85,500 employees as of March 31, 2010, we combine a unique global delivery model infused with a distinct culture of customer satisfaction. A member of the NASDAQ-100 Index and S&P 500 Index, Cognizant is a Forbes Global 2000 company and a member of the Fortune 1000 and is ranked among the top information technology companies in BusinessWeek's Hot Growth and Top 50 Performers listings.

Start Today

For more information on how to drive your business results with Cognizant, contact us at inquiry@cognizant.com or visit our website at www.cognizant.com.

World Headquarters
500 Frank W. Burr Blvd.
Teaneck, NJ 07666 USA
Phone: +1 201 801 0233
Fax: +1 201 801 0243
Toll Free: +1 888 937 3277
Email: inquiry@cognizant.com

European Headquarters
Haymarket House
28-29 Haymarket
London SW1Y 4SP UK
Phone: +44 (0) 20 7321 4888
Fax: +44 (0) 20 7321 4890
Email: infouk@cognizant.com

India Operations Headquarters
#5/535, Old Mahabalipuram Road
Okkiyam Pettai, Thoraipakkam
Chennai, 600 096 India
Phone: +91 (0) 44 4209 6000
Fax: +91 (0) 44 4209 6060
Email: inquiryindia@cognizant.com