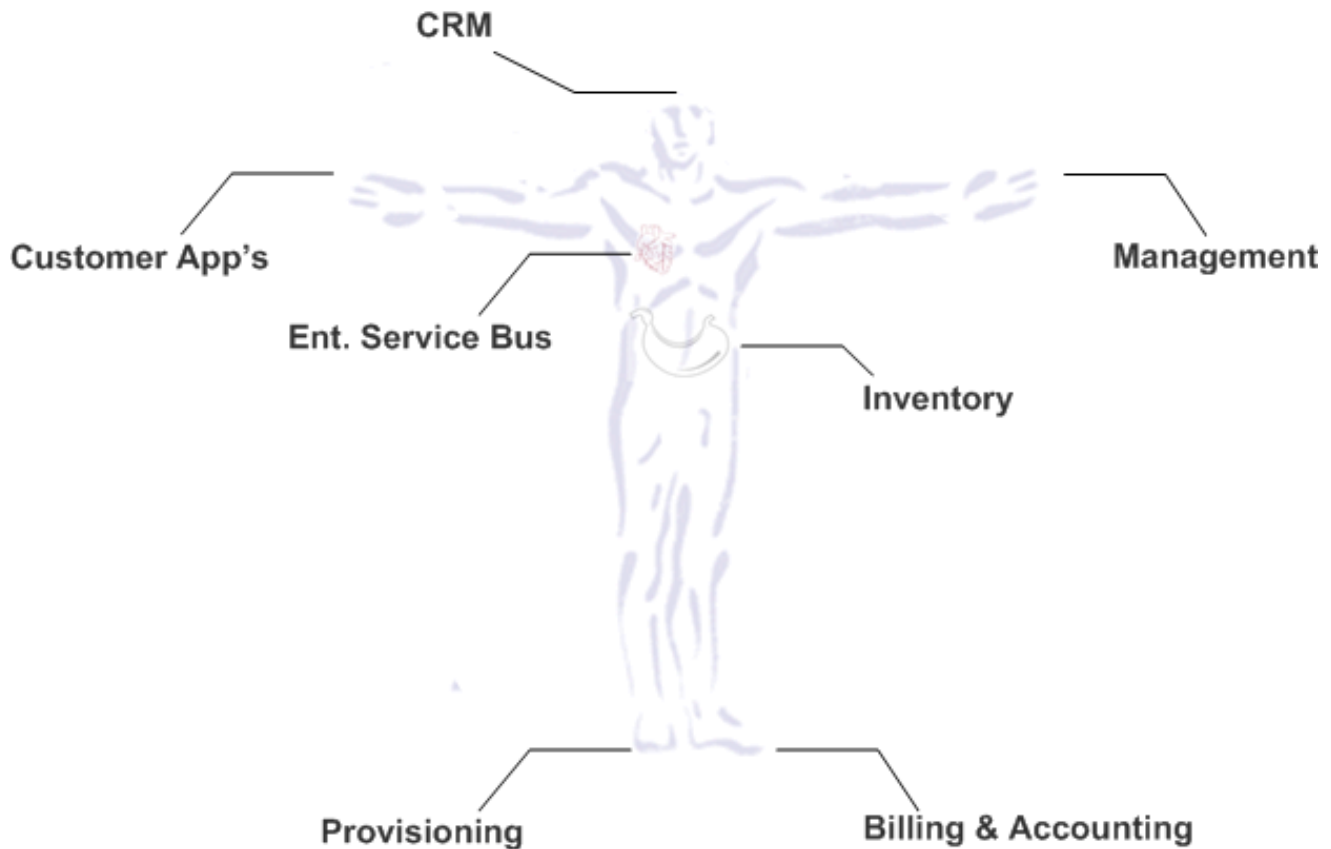


# Trends & Best Practices: IT Systems Development



Leonid Consulting

August 2008

V1.1

Contents

---

- Contents.....2
- Introduction .....3
  - Should I read this?.....3
  - How should I read this?.....3
- 1 Trends & Critical Success Factors.....4
  - 1.1 What's changed?.....4
  - 1.2 How has systems development changed? .....6
  - 1.3 How does the Enterprise Service Bus approach affect my economics? .....8
  - 1.4 How does a project like this need to be managed to be successful? .....9
- 2 Foundation Concept: the Model-View-Controller Framework.....10
  - 2.1 Haven't we had enough "frameworks"? What is this? .....10
  - 2.2 What is MVC? .....10
  - 2.3 Model.....11
  - 2.4 View.....11
  - 2.5 Controller.....11
  - 2.6 Why MVC?.....12
    - 2.6.1 Team member specialization and focus.....12
    - 2.6.2 Parallelization of development tracks .....12
    - 2.6.3 Enhanced testability.....12
    - 2.6.4 Code is future-proofed against technology shifts.....12
  - 2.7 MVC and Loki .....12
- 3 The Anatomical Model of Systems Development .....14
  - 3.1 What is the "Anatomical Model"? .....14
  - 3.2 How does the Anatomical Model relate to the Model-View-Controller framework? .....16
  - 3.3 How does the MVC "Model" operate across the Anatomical Model?.....19
  - 3.4 How does the MVC "View" operate across the Anatomical Model? .....20
  - 3.5 How does the MVC "Controller" operate across the Anatomical Model?.....21
- 4 Leonid's Approach.....23
  - 4.1 CRM: Loki & SugarCRM .....23
    - 4.1.1 Why Loki for CRM? .....23
    - 4.1.2 Why Sugar CRM?.....23
  - 4.2 Customer Applications .....25
    - 4.2.1 Why BroadWorks for call processing? .....25
    - 4.2.2 Why Loki Portals™ for portal development?.....26
  - 4.3 Management.....27
  - 4.4 Enterprise Service Bus.....27
    - 4.4.1 Why Loki™ for Enterprise Service Bus? .....27
    - 4.4.2 Why Microsoft BizTalk™ for Enterprise Service Bus?.....27
  - 4.5 Prometheus.....27
    - 4.5.1 Why Prometheus™ for Inventory? .....27
  - 4.6 Provisioning.....28
    - 4.6.1 Why Loki™ for provisioning? .....28
  - 4.7 Billing & Accounting .....28
  - 4.8 How about an example? .....28
- 5 References.....30

## Introduction

---

### Should I read this?

This paper explains two current IT/systems frameworks in the context of an IP communications services business. If you are a

- Manager: it will provide you useful tools for IT planning, and project management.
- A Buyer (or otherwise responsible for vendor and project evaluations): it will give you a framework to organize your thinking about what building blocks you need, the requirements you should place on them, and how to compare and standardize them.
- Designer or engineer: you'll have a chance to review explanations of one framework where you're probably at least somewhat familiar and another where you probably are not. It will also give you a systematic overview of one approach to the construction of a best practice IT topology.

### How should I read this?

The first section "Trends & Critical Success Factors" reviews major industry trends and how they're affecting the development of IT systems in the IP communications service provider space. If you're interested in how the industry at large is evolving and how that affects service providers, this section is for you. The second section briefly reviews the MVC (Model-View-Controller) framework, an important pattern for modern systems development and a building block of the rest of the paper. Many with a development background will be familiar with this framework; for those unfamiliar with MVC, the section is intended for a general audience. The third sections sets out an "anatomical" framework for systems development, reviewing all the components and how they should best fit together. If you're grappling with the cost, complexity, organization, or functionality of your systems, this section will provide useful organizing ideas. The final section explains Leonid's approach to systems development, using the anatomical framework. If you're interested in working with Leonid and/or a more detailed example from the previous section, this last section is worth reading.

## 1 Trends & Critical Success Factors

---

### 1.1 What's changed?

More aggressive competition and better/cheaper software are two key forces successful carriers are mastering. The first is due to convergence and deregulation, and the second to do with standardization and price/performance improvements in the IT industry as a whole. The first makes life harder, the second makes it easier. A solid competitive strategy will get you through the first, and a well considered, modern IT program will allow you to exploit the second to drive cost out of your business.

Michael Porter's Five Forces<sup>1</sup> framework is probably the easiest way to summarize the effect of these on the business. The framework analyzes the state of an industry sector by looking at five key forces:

#### 1. Bargaining Power of Buyers

In our case, these are consumers and business that buy communications services. They now have more and higher quality choices for service.

**Key drivers for your systems:** design for high-quality self-care interaction and get cost out of the business because prices continue to drift downward.

#### 2. Bargaining Power of Suppliers

Better news here: software is better and cheaper, too. Open source has provided high quality building blocks your suppliers can use for free, global development teams have improved their economics, and consolidation has often meant the emergence of multi-industry best of breed packages that are priced competitively for the market at large. These are offered by the likes of Oracle, Microsoft, SAP, etc.

**Key drivers for your systems:** look for systems built on open source and/or industry-leading components. Require good, clean interfaces (see next sections) and discrete building blocks. Be wary of large monolithic systems built by telco specialists as these are often built on outdated foundations with a very high cost structure.

#### 3. Threat of New Entrants

The separation of applications from transport is bringing more players and specialty players into the market.

**Key drivers for your systems:** make sure you have a clear competitive strategy and an IT system to match. See "Trends & Best Practices: VoIP Provisioning" and "VoIP Service Development: Closing the Loop with Prometheus" for more ideas in this area. Both of these are available for free on [www.leonidconsulting.com/publications](http://www.leonidconsulting.com/publications).

#### 4. Threat of Substitutes

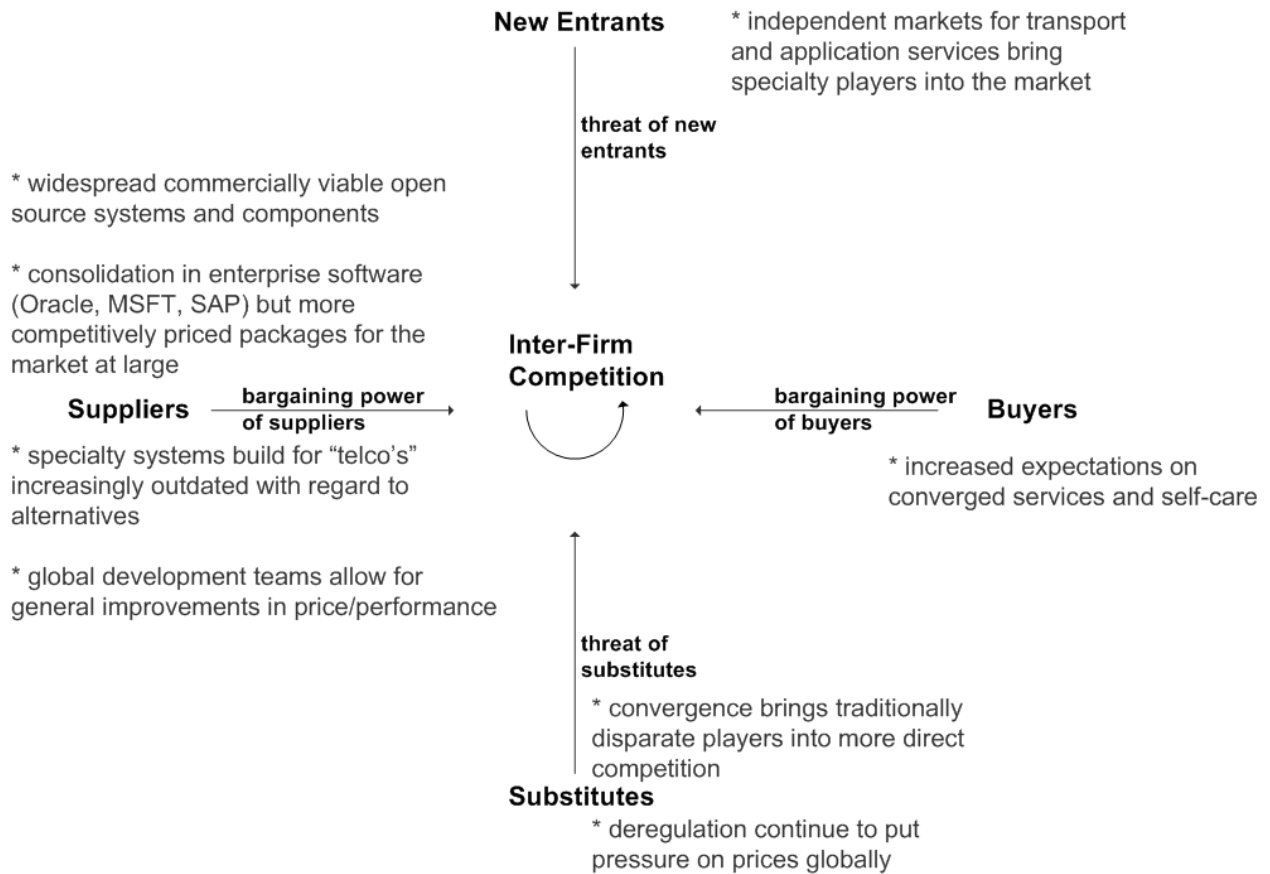
Convergence and deregulation have substantially increased the rate of competitive substitutions. The implications here are largely the same as item #3 above.

#### 5. Inter-Firm Competition

Convergence and deregulation have also blurred this item vs. #4: who is a direct vs. indirect competitor?

The diagram below summarizes these items:

Figure 1 Five Forces Analysis of Communications Service Provider Space



## 1.2 How has systems development changed?

The principal change is a move toward more tightly integrated data and systems in order to support converged services and real-time provisioning events like customer self-care. Investments in data and process integration will also serve to drive cost out of the operation. This approach contrasts with legacy systems in that the legacy systems tend to be “siloes” and operated separately. While the data is interdependent, they are kept largely independent. Variations on the legacy approach are:

- a. provisioning scripts to add accounts, etc. in a single action
- b. maintaining a consolidated subset of this data mediated into a “mainframe” database

The problem with the first variation is that while it may do the job of adding the accounts, reporting, changes, deletions, and reconciliation are difficult since the data is not truly integrated. The problem with this second approach is that this master data store must be kept current, which is often difficult with complex subsystems, and as the subsystems’ data models change, the master data store must be updated before new features can be brought online. While a centralized database may be necessary for certain types of systems, the operator pays a heavy price to reach a fully functional mediated data store of this type, and it should be avoided where possible. The HSS in the IMS architecture is a notable example, especially given the challenges to date of rolling out a fully functional multi-vendor HSS.

In an Enterprise Service Bus, there is an established master instance of every important customer attribute (ID, phone number, device definition, etc.). All other instances of this data are made dependent and are updated either through mediation (where updates are pushed to the dependent subsystem) or federation (where the subsystems are made aware of the location of the master instance of the applicable information). While there is a single view into the service bus, the various subsystems may be accessed and operated directly, as long as all relevant attributes are changed by way of the master instance on the service bus. In the case of discrepancies, audit and reconciliation is simpler since the master instance of all data is established in the design.

The Service Bus itself handles the modelling of the various business processes and transactions as well as messaging between the systems, implementing the various interfaces of the subsystems: SOAP/XML, stored procedures, scripts, etc. Leonid’s Loki™ and Microsoft’s BizTalk™ are both Service Bus applications and we’ll review these in more detail in sections 3 & 4.

The following on the next page contrasts the approaches:

**Legacy System**

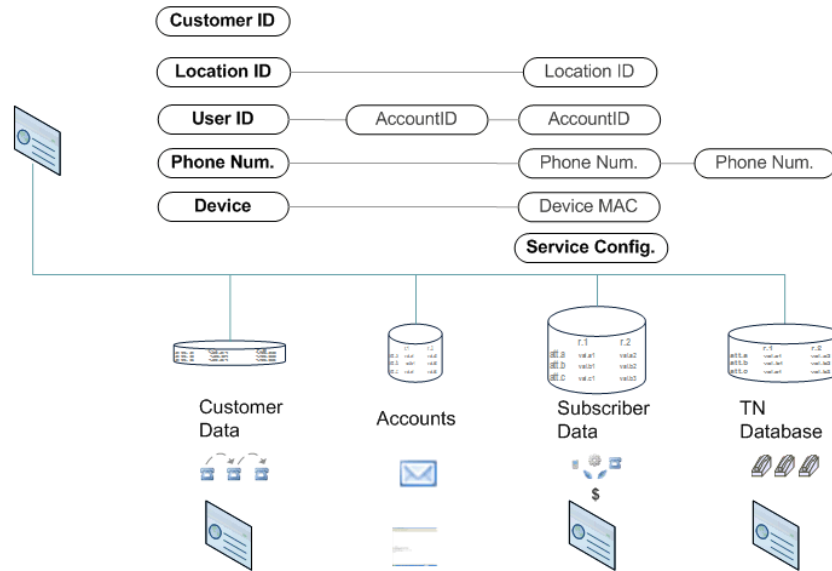
**Views**

**App. Logic**

**Data**



**Enterprise Service Bus**



### 1.3 How does the Enterprise Service Bus approach affect my economics?

The table below provides a summary of the differences between an ESB and a traditional OSS on four key economic drivers:

- Risk: How much am I out of pocket before I know this is going to work?
- Capital Cost: This is the cost for software and other infrastructure.
- Operational Cost: This is the cost across both your own internal staff as well as the use of outside parties for implementation. This also includes the soft costs of decreased efficiency during a transition period.
- Functionality: This refers to the ability of the system to deliver against requirements.

Variable	Enterprise Service Bus	Traditional OSS
Risk	Low, and rapidly declining over the course of the implementation. All IT projects have risk, particularly ones that have to do with business processes, which are ultimately human processes. The nice thing about an ESB is that it allows you gracefully phase in functionality. You can get a few core things in place that you really need, make sure it's working for you, and then fold in more subsystems on a success basis.	High and long standing. Most traditional OSS systems require you to reorganize your systems and business processes around them, and that takes a long time. Most have centralized databases that must hold the master instance of data, and once out of synch they're difficult to reconcile. Have you ever heard a sentence beginning with "You have to do that through the...."? Yes, that's the problem. You have to change a lot of things at once and it takes awhile to see if that is going to pan out.
Capital Cost	Moderate and incremental. While you will need some base infrastructure in place to start, using commercial off the shelf components like Microsoft BizTalk (~8.5k USD) keeps costs down and allows you to add new modules as needed on a success basis.	High. While some of these systems have a certain amount of pay-as-you-grow licensing, most require substantial up front payments. The per subscriber cost on these is generally higher as well.
Operational Cost	Moderate. All large IT projects take a substantial amount of elbow grease. The main advantage on operational cost with ESB is that you can spend on a success basis. And to the extent you can use commercial off the shelf components you'll be able to draw from a large pool of talent with generally lower costs.	Somewhat higher. All systems (ESB or legacy) take work to implement. The main disadvantage in terms of OpEx on traditional OSS is that you have to work with specialists on what is generally a proprietary system and you are then captive to their rates.
Functionality	High out of the gate, increasing thereafter. Functionally, you'll get out of the gate much sooner with something you can use.	Good if you can make the system work, but you'll generally be in limbo for three to six quarters. Finally, many of these systems don't ultimately offer the flexibility and performance the providers require.

#### **1.4 How does a project like this need to be managed to be successful?**

An ESB implementation should be a top-down initiative. The reason is that if it is “something IT is doing” you will usually have trouble getting the necessary cooperation across your organization. These types of systems generally represent major changes in the way things are done. In a modern organization, data is the moat around the proverbial organizational silo’s. It’s often challenging to persuade various parts of the organization to cooperate in making their operations more transparent, and that is what an ESB is all about.

This concludes our section on trends and critical success factors. The next section provides an overview explanation of the MVC (Model-View-Controller) framework.

## 2 Foundation Concept: the Model-View-Controller Framework

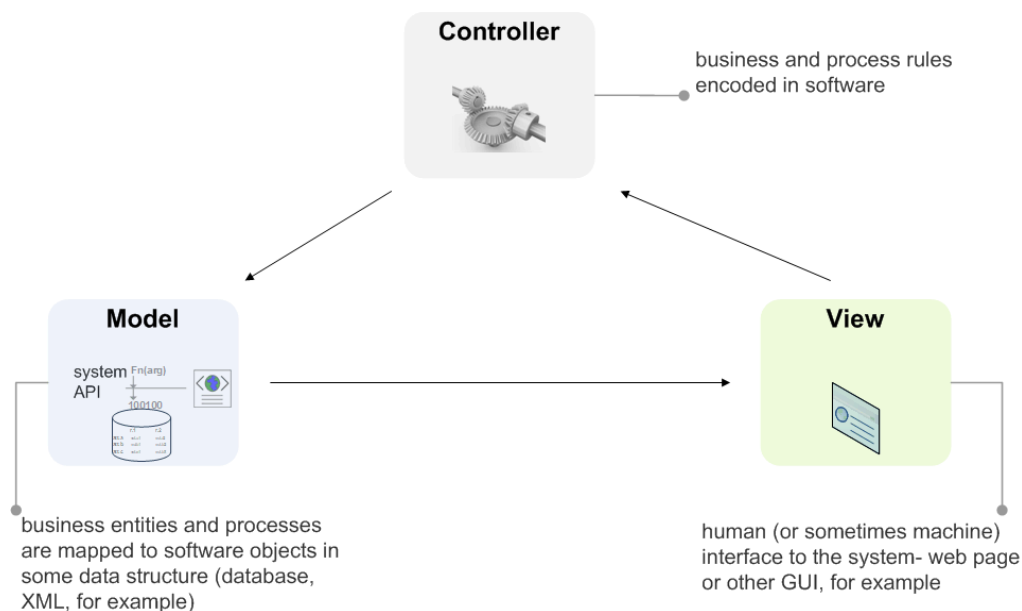
### 2.1 Haven't we had enough "frameworks"? What is this?

The secret truth of software development is this: writing a piece of software to do something is generally not that difficult. The corollary to this secret truth is that bringing order to the unruly art of software development *is* difficult. This is particularly true in IT/systems development where the principal task is itself bringing order and efficiency to a range of (often complex and disparate) systems. The MVC framework has emerged as one of the most prominent working frameworks and it is an essential part of any major development undertaking. In this section we will provide a high level overview of the MVC framework, review its benefits to the development process, and review a brief example.

### 2.2 What is MVC?

MVC is one of the more fundamental development best practices arising out of the past 10 or so years of Internet commercialisation. Its focus is software and project organization. The Model-View-Controller (MVC) framework was developed to separate conceptual layers in the development stack. This has become increasingly important as more organizations realize the value of reusable components and assembly-style development strategies such as Enterprise Service Bus.

The MVC framework provides a separation of concerns that map easily to well-defined team roles and formalizing software "fault lines" where changes in technology strategy and providers can be most easily accommodated. The diagram on the next page provides an overview of the MVC heuristic and the sections that follow detail the components:



## 2.3 Model

The Model layer lies at the heart of all MVC systems. This is the core level at which business requirements are mapped into software data objects. A good model will map as directly as possible to the relevant real life entities involved in the target business process. An overly simplified model will prevent essential details from being tracked and managed, while an overly complex model hinders understanding and efficiency for users as well as developers. Key skill sets for contributors to a Model are:

- broad understanding of the business process in question
- ability to analyse the performance and maintenance impact of alternate design decisions

## 2.4 View

The View layer is where the rubber meets the road for the users of the system. This is another name for the concept of a “presentation tier” – a way for data and processes to be organized and presented to the user. Views encapsulate interface standards (such as names, colors, and window and UI component size and placement) and mediate between the system’s Model and the human interfaces that are ultimately the most important part of any system. Key skill sets for contributors to a View are:

- broad understanding of the business process in question
- formal training in human-computer interaction
  - ability to efficiently map the business processes to end user presentation and operations
  - ability to formalize a visual hierarchy and pattern

## 2.5 Controller

Controllers contain the business logic of the system. They are the place to centralize the automation of business processes so that future updates to process logic are isolated to one area. This is a key concept and driver of modern software architectures – the DRY principle: Don’t Repeat Yourself. Generations of poorly organized, inflexible and bug-prone systems have clearly shown the dramatic benefits of handling one process in only one place in code. As an example, the SOA approach was developed in support of this principle as a way to provide greater access to software components embodying business practices, rather than trying to remodel and rebuild them for use in varying parts of an organizational ecosystem. When a user takes an action in the system, the Controller is the layer at which that activity is translated to operations on the underlying Model. Key skill sets for contributors to a Controller are:

- broad understanding of the business process in question
- formal training in software architecture and algorithms

## 2.6 Why MVC?

The key benefits of employing the MVC framework are as follows:

### 2.6.1 Team member specialization and focus

Each code area often requires different skills, training, and expertise:

1. Database (and/or integration) experts work on the underlying data Model
2. Specialized algorithmic knowledge may be needed to accomplish various actions at the Controller level
3. UI and design experts can independently code and manage the interface via Views

One of the key places where this benefit pays off is when it is hard to find specialists in one or more of the MVC layers. Development can be started in other parts of the system while specialists can be found or trained to backfill the missing area. If all team members needed to know all the necessary skills for the entire project, no work can be started until the staffing challenge is overcome.

### 2.6.2 Parallelization of development tracks

Each layer of an MVC system can be developed and tested independently, assuming their interfaces are appropriately defined. This gives project managers the ability to manage sub-teams discretely on each area.

### 2.6.3 Enhanced testability

One of the less-obvious benefits to the MVC pattern is that it tends to substantially simplify testing. GUI testing, even with today's automated toolsets, remains labor-intensive. By creating the smallest possible software interface layer in the View, unit and integration tests can be written against the Controllers directly, testing the core of the applications logic and supporting systems.

### 2.6.4 Code is future-proofed against technology shifts

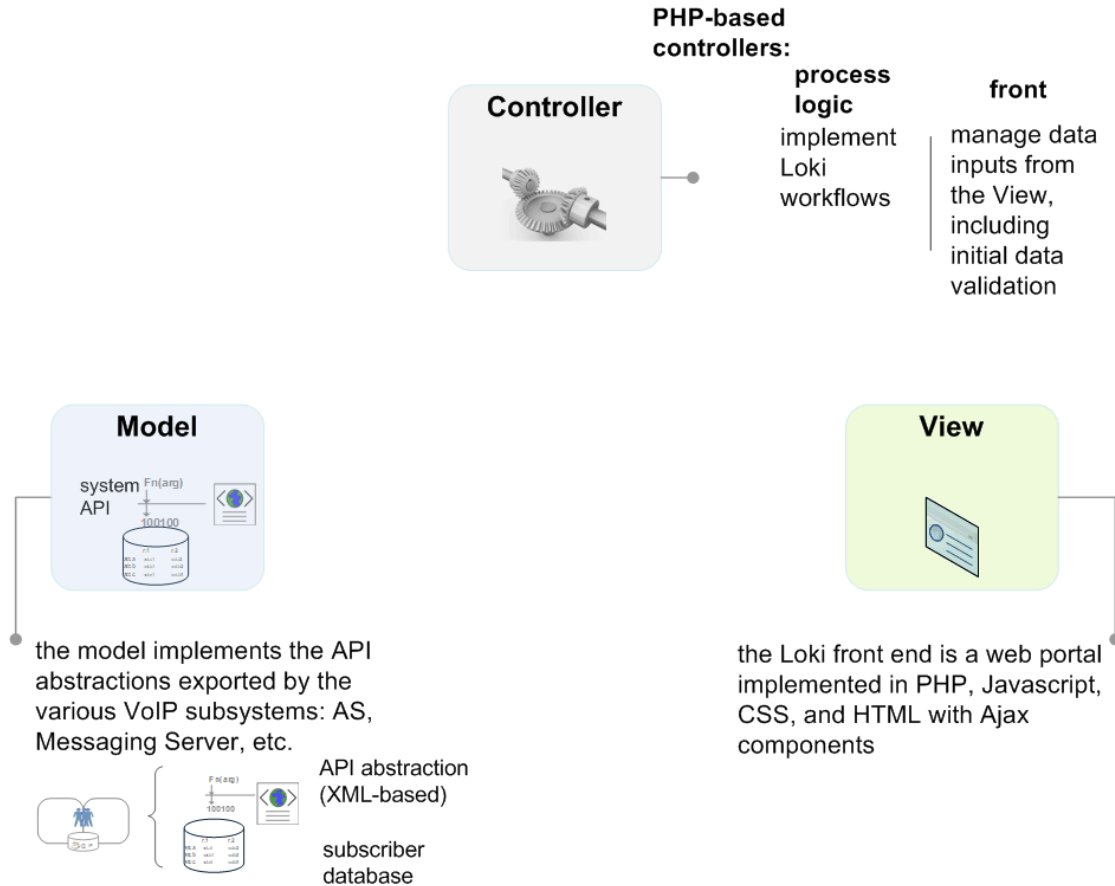
We all realize things change, we just don't always prepare effectively for that change. Today you need a new user interface, while tomorrow having a web services interface is a life-or-death issue. The same thing applies on the database side – moving from a closed to an open-source database, for example, could be a compatibility nightmare without a clean API layer at the Model level. These kinds of problems are easy to resolve with an MVC-based development program.

## 2.7 MVC and Loki

Architectural concerns are equally important in the software product selection process. A system that is designed from the outset to be flexible and maintainable will reward you with fewer bugs, easier, more rapid updates, and can grow and adapt to fit your changing needs as the market evolves.

Leonid's Loki product has been designed from the ground up with forward-thinking Web 2.0 design features such as a MVC architecture, an AJAX<sup>1</sup> user interface, and full ESB-readiness, to provide easy integration with your existing and future back-office and VoIP subsystems. In terms of the MVC we've reviewed here, the current version of Loki is organized as follows:

Figure 2 MVC Loki Example



Element	Role
Model	The Model here is represented by the business objects exposed via APIs from third-party VoIP subsystems. Generally, the subsystems have some kind of an API abstraction to structure interaction and enforce data validation into the database- and in that case the model is the API abstraction. The interface from the Model to the VoIP subsystems is either scripted or structured in XML.
View	The View component is the Loki web-based front end. This is built from PHP, JavaScript, CSS, and HTML with Ajax components.
Controller	There are several sets of controllers. The first are the Front Controllers that are the initial interface to the View. They accept data and perform initial validation of the user data. Subsequently, there are various other controllers that implement the Loki service bus and interface to the various Models.

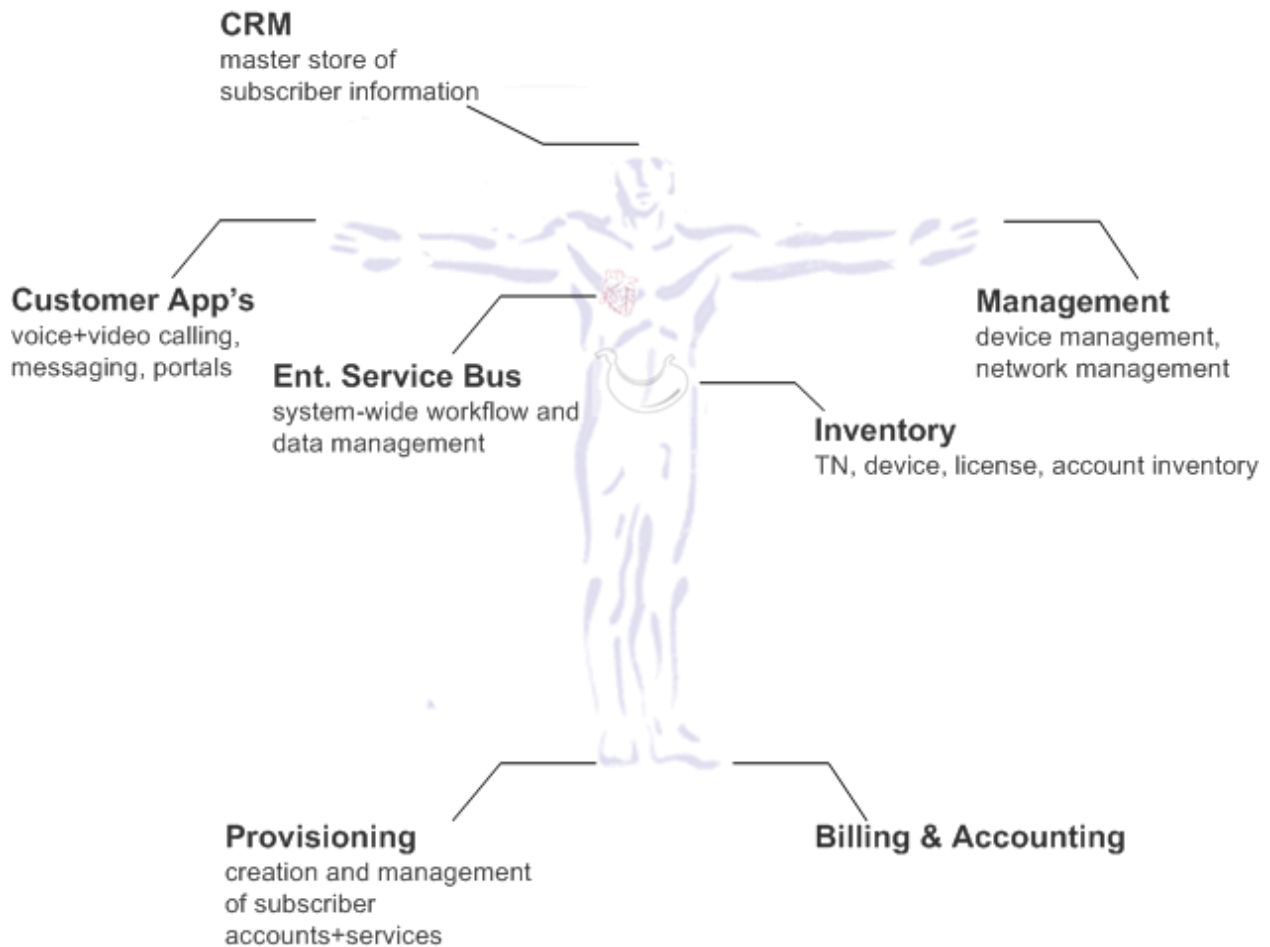
<sup>1</sup> Asynchronous JavaScript and XML – a technique to improve web interfaces by refreshing page components rather than complete pages to respond to user requests. See [http://en.wikipedia.org/wiki/Ajax\\_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming)) for further details.

### 3 The Anatomical Model of Systems Development

---

#### 3.1 What is the “Anatomical Model”?

One of the hardest parts of planning any IT infrastructure is sorting out what everything does, does best, and how to use it. Many vendors contribute to the problem by encouraging a proprietary, vendor-centric “view of the world”. This section seeks to explain a simple, standard model for the major building blocks of a service provider’s IT system for purposes of evaluation, design, planning, and project management. We’ll look at the major components in terms of the human anatomy, something we all more or less understand:



Most of these terms are common and familiar, but it's important to be sure we establish a clear definition:

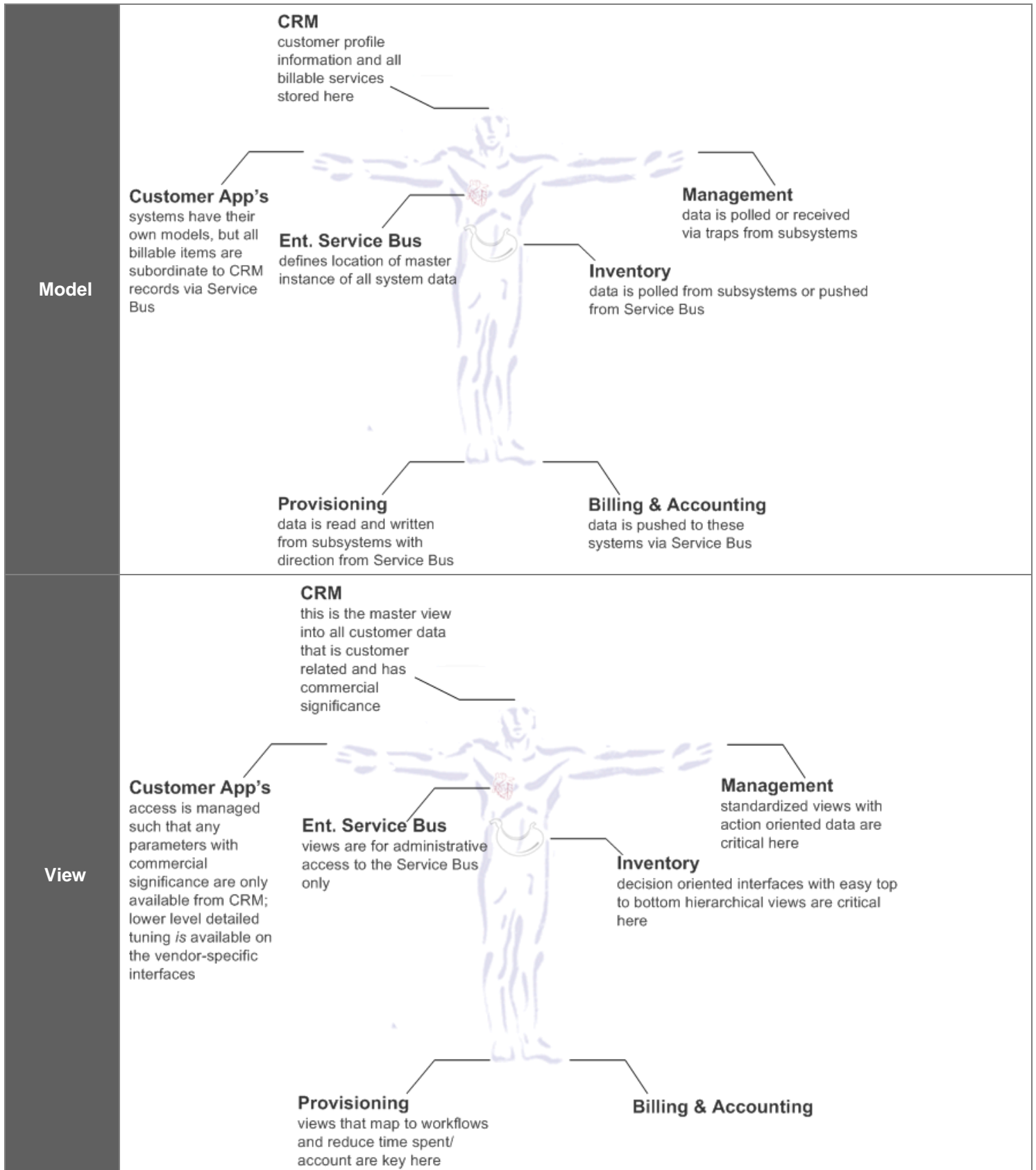
Element	Description
CRM	<p>The Customer Relationship Management system is aptly named- this is the system where anything having to do directly with a customer's service (commercially, at least) should be stored and managed. In the case of a service provider this is generally their plan(s), contact information, and device information.</p> <p>Key functional requirements:</p> <ul style="list-style-type: none"> <li>• ease of use for wide audience</li> <li>• capable of accommodating all relevant customer information: core structures built in, flexibility to easily customize as needed</li> <li>• strong external interfaces for integration into the service bus</li> </ul>
Customer Applications	<p>These are the systems, generally application servers of various types, that provide end users services (voice &amp; video call processing, media, messaging, etc.).</p> <p>The requirements here are many and varied, but have principally to do with the quality of the service you can offer to the customer.</p>
Management	<p>These are the systems that make sure the customer applications and other critical parts of the system are up and running and in good health.</p> <p>Key functional requirements:</p> <ul style="list-style-type: none"> <li>• easily filter and organize critical and non-critical issues</li> <li>• ability to summarize key attributes in homogenous fashion across elements</li> <li>• depth on element specific attributes <ul style="list-style-type: none"> <li>○ advanced diagnostics, release levels, patch levels, etc.</li> </ul> </li> </ul>
Enterprise Service Bus	<p>We've reviewed the enterprise service bus concept in section 1. The ESB is how we organize inter-system transactions and data around our processes.</p> <p>Key functional requirements:</p> <ul style="list-style-type: none"> <li>• simple interface for creating and managing business processes, ideally directly accessible by non-technical personnel/business process owners</li> <li>• breadth and depth of interface adapters <ul style="list-style-type: none"> <li>○ web services/XML, stored procedures/SQL, SMTP are the basics</li> </ul> </li> <li>• transparent reporting framework for viewing the state of a process</li> </ul>
Inventory	<p>We need to know what we have- devices, phone numbers, service plans/licenses, and we need to know that across all the various systems.</p> <p>Key functional requirements:</p> <ul style="list-style-type: none"> <li>• ability to acquire data from target sources</li> <li>• ability to track and model linkages and hierarchies between systems and elements</li> <li>• ability to audit above in fashion that leads to clear, simple rectification of issues</li> </ul>
Provisioning	<p>This is the system we use to initiate move's, add's, changes and deletes across subsystems.</p> <p>Key functional requirements:</p> <ul style="list-style-type: none"> <li>• ability to easily model relevant production configuration standards</li> <li>• simple interface for add's, changes</li> <li>• robust &amp; descriptive error handling</li> </ul>
Billing & Accounting	<p>This is where the charges are calculated and payments debited and tracked.</p> <p>Key functional requirements:</p> <ul style="list-style-type: none"> <li>• simple modelling of relevant product and account standards (account hierarchies and</li> </ul>

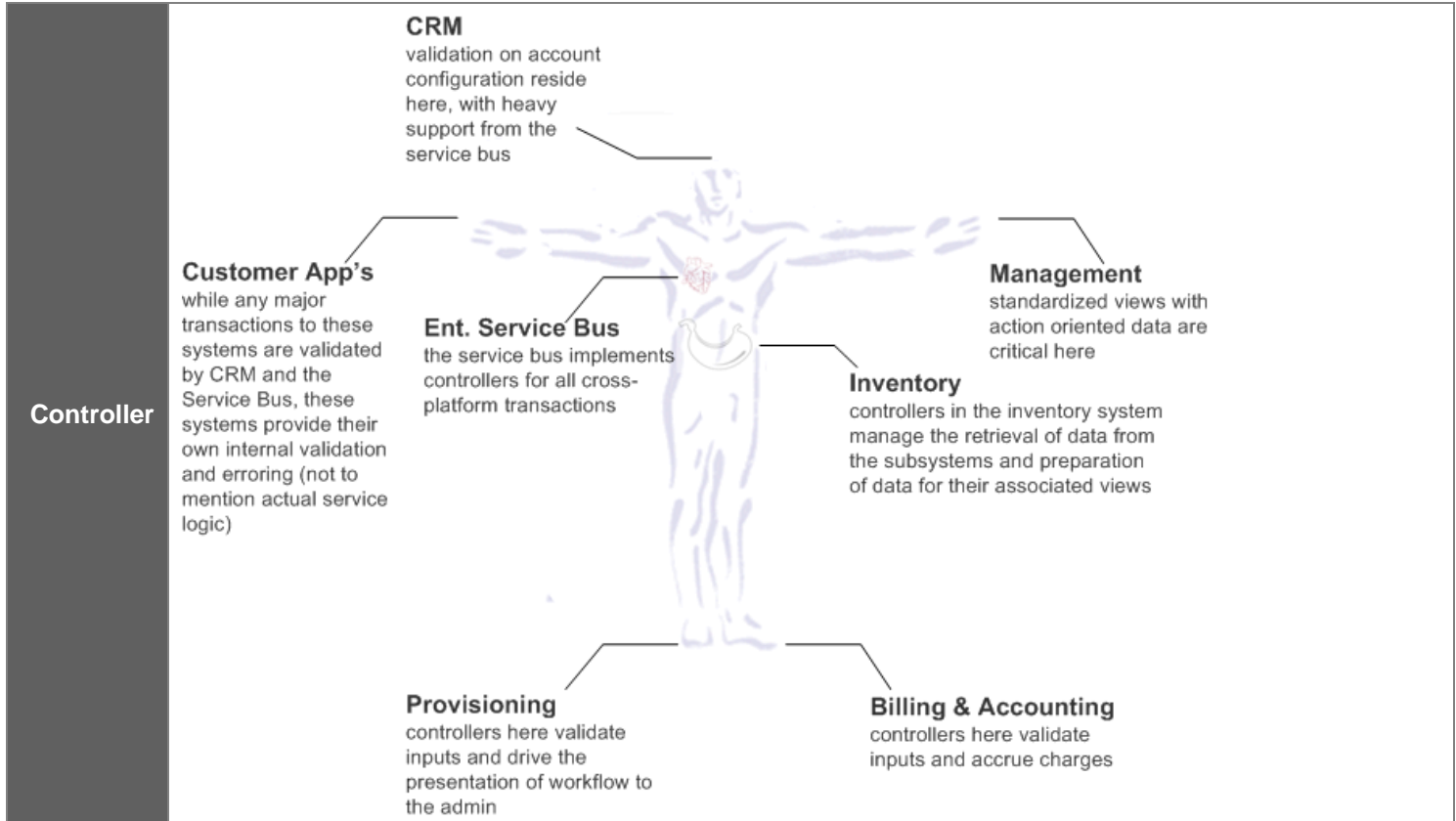
	<p>product pricing models)</p> <ul style="list-style-type: none"><li>• ability to independently rate and update customer plans</li><li>• transparent reporting and flexibly invoice generation</li><li>• interface to GL/accounting</li></ul>
--	---

**3.2 How does the Anatomical Model relate to the Model-View-Controller framework?**

Excellent question!

For starters, there's a summary on the next page:





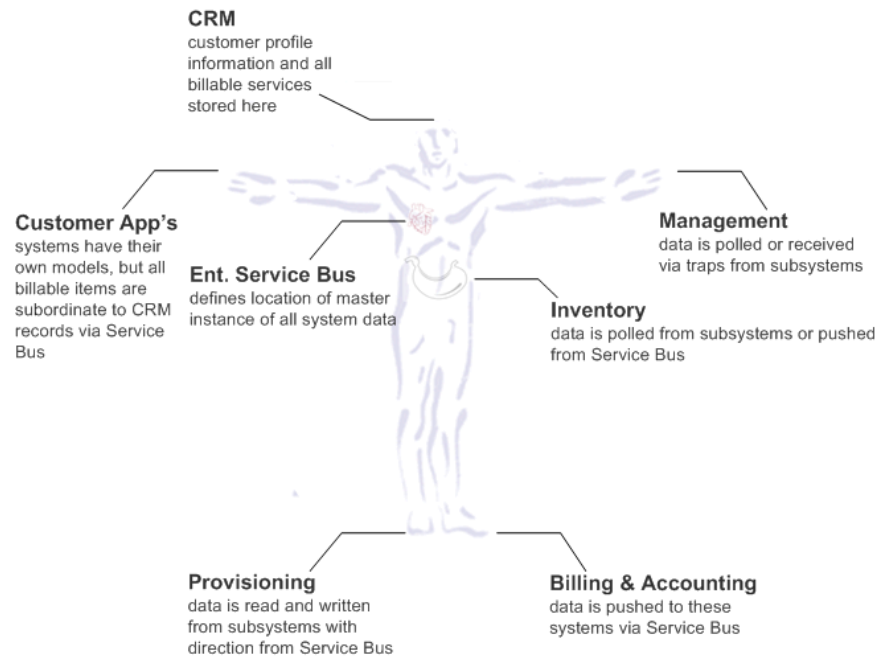
### 3.3 How does the MVC “Model” operate across the Anatomical Model?

The Model is the IT representation of a business entity- a service plan, a subscription to voice applications, or a simple phone number, for example. Commercially relevant information on all billable items should reside in the CRM. This does *not* mean you may not want, for example, the ability of a CSR to view the state of a subscriber’s account (call forward, etc.) when they log into CRM- but that is the job of the

View and a Controller that knows how to go out to the Service Bus and acquire the relevant information (from the voice application server, for instance).

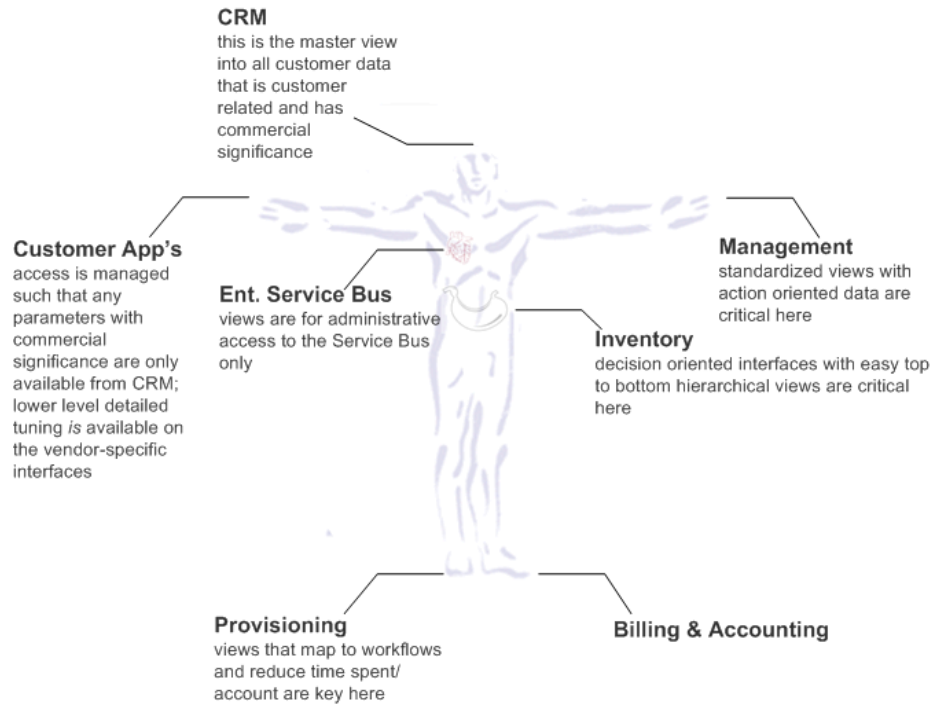
Customer Applications are often very complex in and of themselves. In terms of the Model, all attributes that are billable (phone numbers, number of accounts, service plans, etc.) should be dependent and pushed to the Customer Applications from the CRM via the Enterprise Service Bus. Detailed service attributes (call forward numbers, etc.) will generally reside uniquely

on the Customer Applications: trying to model every service attribute elsewhere and push it to the Customer Applications is generally a recipe for failure. Other than configuration information, information on the Management systems are generally polled and/or received via traps from other subsystems. The Enterprise Service Bus is the model of the models, so to speak. However, it’s best to avoid storing much data here- the job of the bus is to know where data resides. Inventory systems, like Management, generally poll data from other sources or have it pushed to them by the Bus. A good Provisioning system has very little of its own data. It primarily provides Views and Controllers against the Service Bus to automated provisioning processes. As we mentioned earlier, trying to build a provisioning system that somehow maintains a “mainframe” database of all other system attributes is rarely successfully, and unnecessary with an ESB-oriented architecture.



### 3.4 How does the MVC “View” operate across the Anatomical Model?

The first question about the view is, “Who is the audience and what do they want to do?” The job of the View is to help them do what they want to do. When you’re designing a system, the View component is often a good place to start. We’ve provided the table below to distil the key elements of each component’s view:

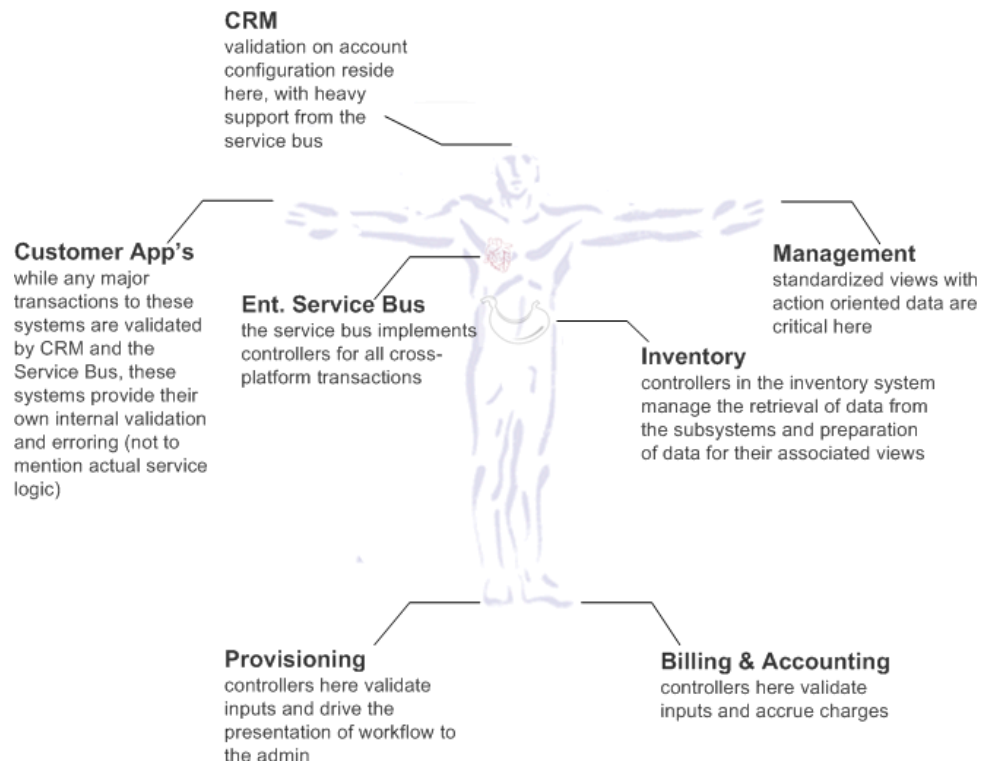


Element	Audience & Objective	Notes
CRM	Audience: customer-facing care representatives and account managers. The chief job of the CRM interface is to help this part of your organization acquire and retain accounts.	Use the flexibility of the CRM's View to build a dashboard that allows for presentation of the necessary information to add and service accounts.
Customer Applications	Audience: end users The objective of these users, of course, depends on their application, but they all have one thing in common: they're trying to use your service to go about their daily business/lives. It's your job to fit in.	The more elaborate Customer Applications will often come with their own interface. However, with convergence, brand identity, and self service being as important as they are, many providers build custom portals to provide a view of Views across services relevant to their particular customers' relationship to the service(s) offered. Executing well here is a critical competitive advantage for many providers.
Management	Audience: NOC and upper level engineers These users are trying to answer the following questions: a) "Is an element malfunctioning?" and b) "If so, why, and what should I do about it?"	A good Management View provides the answers quickly and easily.
Enterprise Service Bus	Audience: business process owners and IT administrators These admin's are trying to model business	A good View is something you should look for as part of the Service Bus application itself. Ideally it provides a simple enough

	processes.	interface to allow business process owners to do at least preliminary creation and modification of business process updates.
Inventory	Audience: management, accounting, internal audit These users are trying to determine where resources (devices, numbers, service plans) are allocated, generally to make management decisions and/or resolve discrepancies.	A good inventory view is oriented towards directly answering the various questions management is trying to answer.
Provisioning	Audience: CSR's, customer project managers These folks are trying to add or modify customer accounts. The measure of success for the View here is the time required and accuracy obtained in provisioning, modifying, and verifying accounts.	Reducing clicks and keystrokes is a big deal here.
Billing & Accounting	Audience: customer project management, CSR's, and accounting Billing parallels provisioning: these users are trying to provision, modify, and delete plans. In a perfect world, this happens automatically.	The less you're looking at this View, the better your system is working.

### 3.5 How does the MVC "Controller" operate across the Anatomical Model?

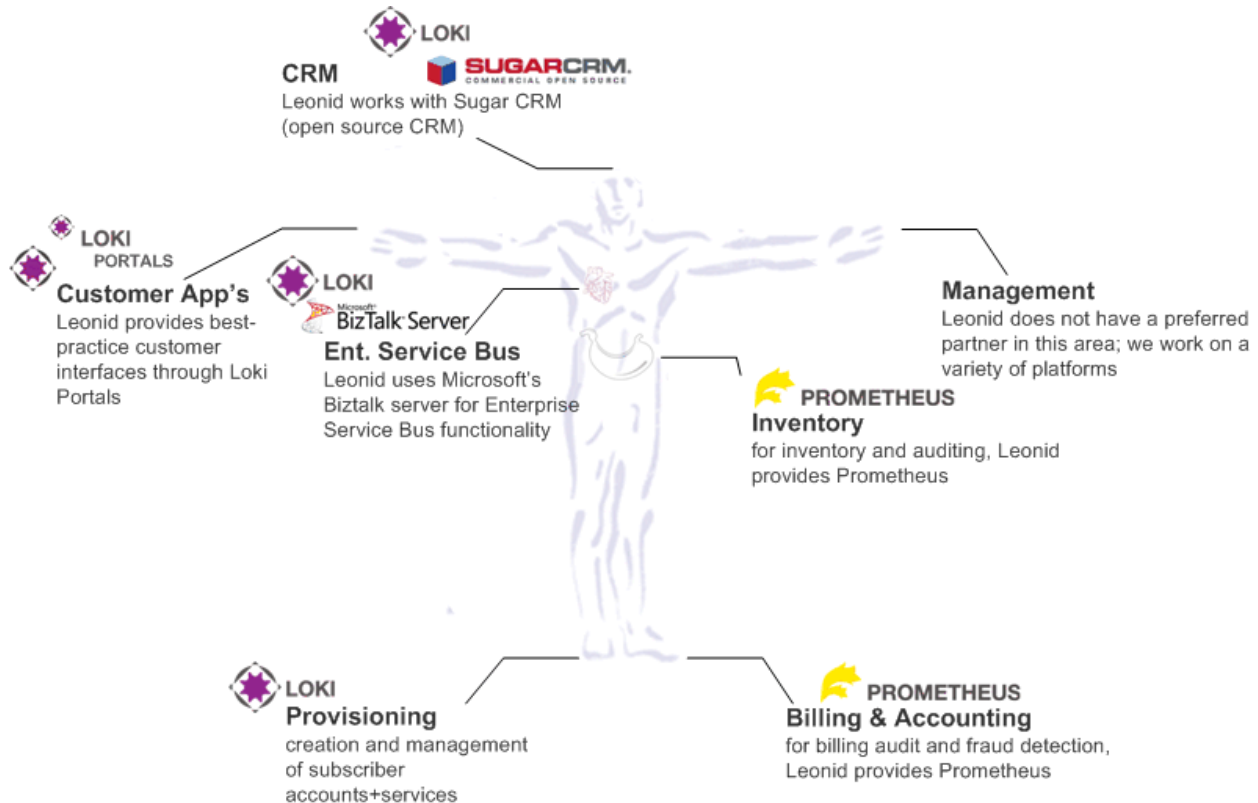
A good controller is like a good assistant or partner: it knows what you're trying to do before you even do it, and helps you remember things you forgot, or, better yet, reduces the amount of things you have to think about at all. In the CRM, controllers are generally "front controllers" calling the Service Bus to show Views from multiple data sources. Customer Applications implement a wide variety of controllers to implement communications protocols and application logic, as well as managing their own internal account structures. The Management controllers mediate data pushed and pulled from the system to present consolidated,



homogenized data to the Views. The Service Bus controller is probably the most intricate, choreographing the interaction of data and processes across systems. The Inventory controller orchestrates the retrieval and organization of data for presentation in their system's Views. Provisioning systems provide an abstraction layer over the various subsystems to automate standard account addition and modify events, and these controllers manage the abstraction. The Billing & Accounting controllers provide data validation and implement the logic to accrue and calculate charges.

## 4 Leonid's Approach

This section describes Leonid's particular approach to implementing carrier IT systems on an Enterprise Service Bus. The following diagram summarizes Leonid's preferred tool set:



### 4.1 CRM: Loki & SugarCRM

Depending on the scale of the project, Leonid uses Loki™ or SugarCRM™. Loki implements a basic service bus allowing it to use a secondary data store (usually the VoIP application server) as the customer database for basic CRM functions. For larger scale projects where broader functionality is required, Leonid uses SugarCRM™, a commercial open source CRM application.

#### 4.1.1 Why Loki for CRM?

Loki provides a basic service bus which allows for the easy composition of a very basic CRM function. If you haven't identified the need for large scale CRM functionality, Loki™ will allow you to engineer basic functions like account status and troubleshooting for little money in a short period of time.

#### 4.1.2 Why Sugar CRM?

Like most enterprise software packages, there is no single attribute of SugarCRM that makes it a hands-down obvious choice. On cost, SugarCRM™ is economical and if you're really on a budget they offer a community edition. The availability of the community edition also allows you to substantially try before you buy, an attribute of the software vendor/buyer relationship we think is important. The out of the box experience is good to very good- the application is easy to install,

customize and begin using right away. Our own (relatively simple) set up took four and a half hours. Finally, the application integration environment is excellent- the process of developing plug-in's and integration is very good. This is largely due to the transparency of the application.

## 4.2 Customer Applications

A full review of all the application a provider might offer and how to do that is, of course, outside the scope of this paper.<sup>2</sup> Here we'll briefly cover call processing and customer portals.

### 4.2.1 Why BroadWorks for call processing?

Assuming your goal is to offer hosted/network call processing in a multi-vendor environment, BroadWorks™ is a good way to go. The vendor space in this area has matured over the last few years, and BroadSoft has emerged as the clear leader in the multi-vendor/standards-based hosted communications space. While there are substantial offerings from legacy vendors like Nortel and Lucent, these tend toward the traditional paradigm of single-vendor monolithic systems. Interoperability is challenging, flexibility is limited.

BroadSoft operates a good interoperability program where they publish “partner configuration guides” for each validated element (IP phones, gateways, etc.). The feature set and overall solution composition is solid and all system functions on the application server are available via a modern XML-based API (their own JSP-based web portal and CLI are implemented on this same interface). Finally, the solution is highly scalable with several major providers now operating large scale networks on the platform.

---

<sup>2</sup> Once we go over 40 pages, our audience starts to dry up.

## 4.2.2 Why Loki Portals™ for portal development?

When we first began to seriously engage in the area of portal development, our experience was that providers wanted something that

- was quick to deploy
- made use of modern interface navigation paradigms
- was easy to integrate into their existing infrastructure
- would not create headaches during upgrades to the underlying subsystems where the portal would interface (VoIP application server, etc.)
- provided an interface to user as well as site admin functions
- provided for call reporting

With these things in mind we built Loki Portals. Loki Portals is a complete portal for IP communications services, so you don't have to start from square one, but it's also built on an open source toolkit called Smarty™ that allows us to separate the front end and the back end. What this means is that you can use your web front end specialist to do the design and implantation of your portal without worrying about the back end implementation. Additionally, we offer maintenance on Loki Portals so when you upgrade a back end system, like BroadWorks™, you don't have to sweat the upgrade on the portals.

The screenshot displays a web-based administrative interface for 'Loki Portals'. At the top, there is a navigation menu with tabs: 'Company Info', 'Manage Users', 'Group Services', 'Disaster Redirect', and 'Call Details'. Below this, a secondary menu highlights 'Instant Call Group' among other options like 'Auto Attendant', 'Hunt Group', 'Music On Hold', and 'Speed Dial 8'. The main content area is titled 'Settings for Instant Call Group group service'. It features a dropdown menu with the value 'demoicg@voip.demo.net (TN: , Ext: )' and a 'Submit' button. Below this is a section for 'Destination Numbers' containing a list with one entry: '2122021300', which has a 'Delete' button next to it. At the bottom of this section is an 'Add New Number' button.

Copyright © 2008 Leonid Consulting, Inc.

### **4.3 Management**

We have no vendor/application-specific recommendations on management beyond the functional requirements in the previous section.

### **4.4 Enterprise Service Bus**

#### **4.4.1 Why Loki™ for Enterprise Service Bus?**

Loki implements a basic VoIP-specific service bus with interfaces to call processing, messaging, and billing. Loki is a fast, inexpensive way to roll out service on a modern architecture. While we will likely add more facility to the Loki service bus over time, for large scale deployments we recommend Microsoft's BizTalk™.

#### **4.4.2 Why Microsoft BizTalk™ for Enterprise Service Bus?**

We've found Microsoft's BizTalk™ Server 2006 an excellent service bus product. At today's price for the Standard edition, \$8,499/host (two processors), it won't break the bank. It is widely deployed across industries (not just communications) and has millions of dollars in R&D behind it every year. In terms of specific functionality, it is probably the graphical interfaces to business process management that are the most notable. The reason these are so important is that these allow managers to make the Controller logic for business processes accessible to the actual business process owners, rather than just technical specialists. This increases the success rate of implementations and cuts down on non-value added time communicating between "business"-oriented staff and technical staff. In general, as a service bus it operates on a solid foundation- easy access to XML, databases, and protocols such as stored procedures and SOAP.

### **4.5 Prometheus**

#### **4.5.1 Why Prometheus™ for Inventory?**

As a system primarily for managerial processes like audit, billing, and network planning, a good Inventory system should be able to acquire data easily from the relevant systems and present it in a format that makes the relevant decision making and/or business processes easier. As a consultancy, we specifically built Prometheus™ to provide easy answers to questions we consistently found ourselves answering for clients:

- What systems are my users using? How often?
- Do we have fraud?
- Where are all my licenses and how are they being used?
- Where is all my service inventory, TN inventory, and device inventory? Is it being used efficiently? Is it being billed for properly?

Prometheus is built on a modern web services architecture that makes updates and reporting easy. Unlike a lot of the legacy systems we've seen, it handles physical assets and non-physical assets (licenses, services, etc.) with equal ease.

## 4.6 Provisioning

### 4.6.1 Why Loki™ for provisioning?

We like Loki™ primarily because it is built on top of a service bus, rather than a (fragile) monolithic, mediated data store, and because it was built alongside people whose job it is to provision and maintain IP communications services. We built Loki™ for them because we just didn't see anything else in the market that was doing the job well. In simple terms, a provisioning system should be evaluated around the following:

- How much does it cost?
- How much time will it save my people once it's in place?
- How much will it increase the accuracy of the orders I provision?
- How hard will it be to get it in place and start using it?
- How much will it cost to maintain it over time?
- How flexible is it with regard to new architectures I'm considering?

We were so disappointed by the answers over the course of our experience with clients that we built Loki™. Loki is inexpensive and easy to put in place and start using right away. It is extremely efficient in its use of a service bus architecture to minimize secondary/mediated data. We carefully examined the drivers of time spent and accuracy obtained on VoIP orders and Loki's front end and controllers are specifically designed to minimize time and maximize accuracy. Finally, you can use Loki as a provisioning front end and/or you can use Loki Web Services as provisioning middleware to reduce the complexity of interfaces on service bus systems like BizTalk.

## 4.7 Billing & Accounting

We have no vendor/application-specific recommendations on billing & accounting beyond the functional requirements in the previous section.

## 4.8 How about an example?

The diagram on the following page shows an example of a basic service bus architecture executing an order. The top section of the diagram shows the available human interfaces to these systems. The application/controller row shows the various applications and the sample workflow. The workflow is simplified and for heuristic purposes only. For instances, no responses are shown in the diagram. The last section shows the data model and the relationships between the systems for referential integrity.

This scenario shows a new order being created, assuming that the necessary entities have already been created in the appropriate systems. The order is created in CRM and then CRM routes a request for order processing to BizTalk. (1). BizTalk then creates the necessary service plan entries in Billing (2). BizTalk then calls Loki™ which handles the provisioning and set up of the various entities in the VoIP application server and message store (4). Finally, BizTalk places order requests and updates tables in the TN and Circuit management systems (5 & 6).



## 5 References

---

<sup>1</sup> Porter, M.E. (1980) *Competitive Strategy*, The Free Press, New York, 1980.