Torry Harris

Distributed e-enterprise **Computing**

The ASPizer™ Toolkit: ASP in a Flash

Authors: A. Ghosh, S. Grandhi, R. Mathew, A. Natarajan and S. Ramaswamy

Intended Audience: This paper is primarily intended for ISVs, AIPs/HSPs and Aggregators that are building and/or hosting applications based on the Application Service Provider business model.

Further Inquiries and Feedback: aspizer@thbs.com

Notices:

Torry Harris Business Solutions
1090 King Georges Post Rd
Suite 103
Edison NJ 08837
USA

1 Creating ASP Offerings

The ASP Model is essentially a new method of delivery of applications to end-users. Until the advent of the ASP model, in order to use an application, organizations had to purchase the applications and necessary hardware and then hire and train personnel on the operations and maintenance of this infrastructure before end-users could use the application. With the ASP model, organizations have to just purchase access to the application from a Service Provider and their end-users can immediately use the application from their browsers; a process that likely takes a few minutes and does not involve any up front infrastructure costs.

At the heart of any ASP offering is one or more web-enabled applications augmented by logic dealing with making the applications available under the ASP model. At the very least, the ASP model requires that there be logic dealing with the following aspects:

- **Profiles** – storing and using information specific to a user or company that can be used to customize user experience with the application.
- **Provisioning** – creating necessary application resources for each new user or company renting the application.
- **Security** – specifying and enforcing policies that ensure users or companies are authenticated and authorized to access and modify only their own data.
- **Licensing and billing** – specifying and enforcing policies that ensure application use and charge for a user or company is consistent with their contract.

While covering these basic aspects is necessary for creating a basic ASP offering, it is not really sufficient. We feel that a broader set of issues needs to be considered when building an ASP offering. These issues arise from the following observations about the ASP market:

- Given the ASP model mainly targets Small and Medium Businesses (SMBs), it is important to be able to easily create seamlessly integrated solutions for specific vertical industry segments from a set of applications.
- There are three fundamental logical roles played by companies in the ASP market – Application Development, Application Aggregation and Application Hosting. Even though some companies play multiple logical roles, it is highly likely that most companies will play a single logical role that best fits their core competency. This implies companies will likely collaborate to create offerings for end-users.
- Currently, software manufacturers are likely to establish OEM relationships and channel relationships whereby their software is marketed and sold by other parties with some value addition. A similar model will evolve in the ASP marketplace whereby applications will be rented out through multiple market channels each of which will bundle in some additional value in terms of augmenting applications and/or services. This requires the applications to be very flexible and customizable to be able to suit all the requirements of different market channels.

With the above observations in mind, we believe that logic dealing with the following aspects needs to be considered when making an application available under the ASP model:

- **Shared security** – the application will need to support various security models dependent on the specific market channel being used to access it.
- **Shared session** – the application will need to support a session that is shared between all applications. This shared session can be used for passing data between applications to facilitate better integration.
- **Shared profiles** – the application will need to support sharing profile information with other applications. Having a shared profile reduces the amount of information to be stored and maintained as well as makes the user's life simpler.
- **Custom Service Level Agreements (SLAs)** – various application configurations will need to be offered to users and companies to meet their desired price and performance points.

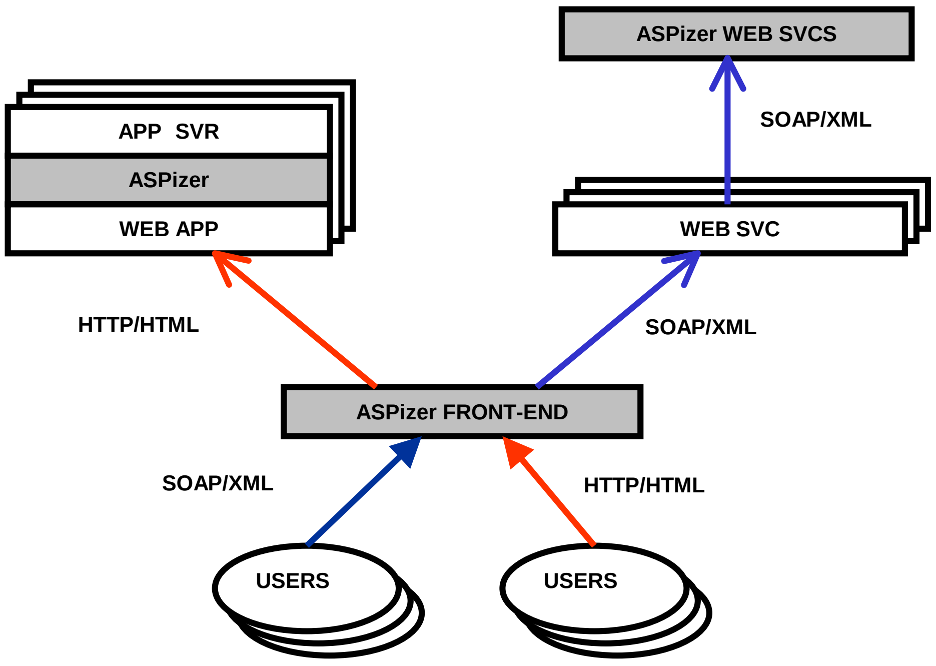
- **Customizable look-and-feel** – the application's look-and-feel will need to conform to the market channel being used to access it. This means the application has to be able to easily support a diverse set of user interface styles.
- **Business process integration** – when multiple applications are offered at an aggregator site, customers may want to integrate the applications or sub-tasks from applications with their own business processes. For instance, a procurement process for a business may involve using a purchasing application, an accounting application and an inventory control application.
- **Cross application functions** – when the application is integrated with other applications at an aggregator site, a single higher level task at the site may involve sub-tasks from multiple applications. For instance, an aggregator site that hosts productivity apps like Human Resources, Accounting and Payroll for small businesses may require a new employee enrollment task to span all three applications so they can all be up to date on the change.

The additional logic needed for the ASP model as mentioned above can be coded into the application itself. While this approach integrates the ASP model well with the application, it has two major disadvantages – increased complexity in the application and decreased flexibility in terms of the application's operating environment. Our ASPizer toolkit provides an alternate approach by providing the application programmer with a set of APIs and a special runtime environment for the application. This approach allows for tight integration of the application with the ASP model as well as reduces the complexity of the application. Furthermore, the application can be isolated from details of its operating environment, thereby providing a large degree of flexibility.

2 The ASPizer Toolkit Solution

The ASPizer toolkit is aimed at organizations involved in the creation of solutions for the ASP market. These organizations could play any logical role – Application Development, Application Aggregation or Application Hosting. The ASPizer toolkit contains features for all these roles.

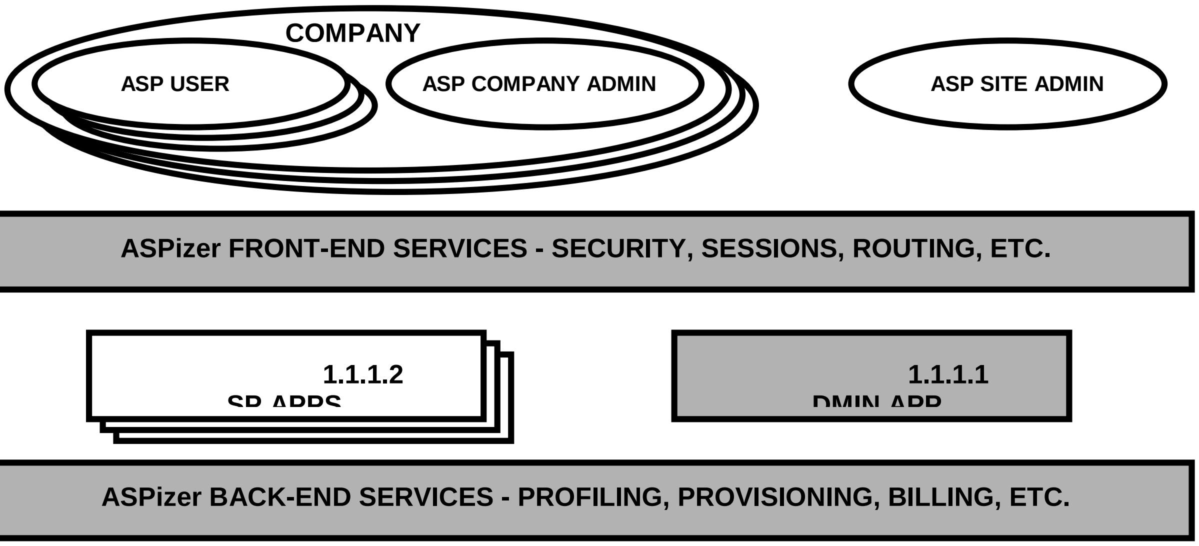
Applications at the core of ASPizer-based offerings could be either web-enabled applications or web services. If the application is web-enabled, we assume it uses an application server technology that adheres to either the Java 2 Platform Enterprise Edition standards or the Microsoft Active Server Pages model. The picture below illustrates how a typical ASPizer-based offering is structured.



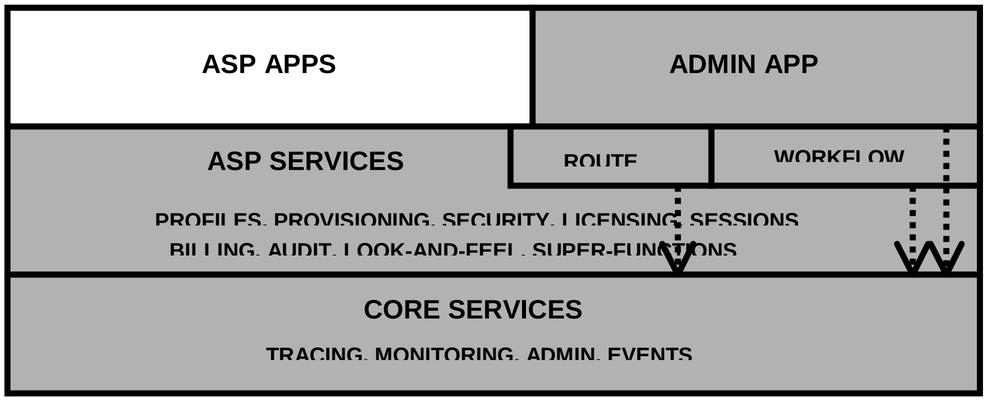
The ASPizer toolkit comprises the following parts:

- The ASPizer deployment tool that accepts an XML descriptor providing details of the nature of ASP mode support desired for the application. The deployment tool uses this information to perform necessary initialization and generates configuration information for use by the runtime.
- The ASPizer runtime that provides all necessary services and support for the application while it is running in the ASP mode.
- The ASPizer APIs that are available for use by the application programmer to add enhanced support if necessary. These APIs are available as method calls within the application server environment or as web services – applications can choose their preferred method of access.
- The ASPizer administration tool for use by the ASP hosting provider for administering all the applications being hosted.

The picture below illustrates a logical view of an ASPizer based ASP offering from a user perspective. ASPizer intercepts user requests and provides a set of front-end services that come between users and the applications they access and use. For example, the security service allows users access only to those applications that they are authorized to use. The applications themselves can use a set of back-end services that provide a lot of the functionality needed for the ASP model. For example, the application can query the security service to find out the role a user is playing and present an application view that is appropriate for that role.


COMPANY
ASP USER
ASP COMPANY ADMIN
ASP SITE ADMIN
ASPizer FRONT-END SERVICES - SECURITY, SESSIONS, ROUTING, ETC.
1.1.1.2
SP APPS
1.1.1.1
DMIN APP
ASPizer BACK-END SERVICES - PROFILING, PROVISIONING, BILLING, ETC.

The ASPizer toolkit services are logically structured as pictured below. There is a layer of core services that are used to build a set of ASP services that ASP applications use. Some of the services like the Router and Workflow services are complex in that they use other services themselves. The Admin application uses the services as well as can be used to configure and monitor the services.


ASP APPS
ADMIN APP
ASP SERVICES
ROUTE
WORKFLOW
PROFILES, PROVISIONING, SECURITY, LICENSING, SESSIONS
BILLING, AUDIT, LOOK-AND-FEEL, SUPER-FUNCTIONS
CORE SERVICES
TRACING, MONITORING, ADMIN, EVENTS

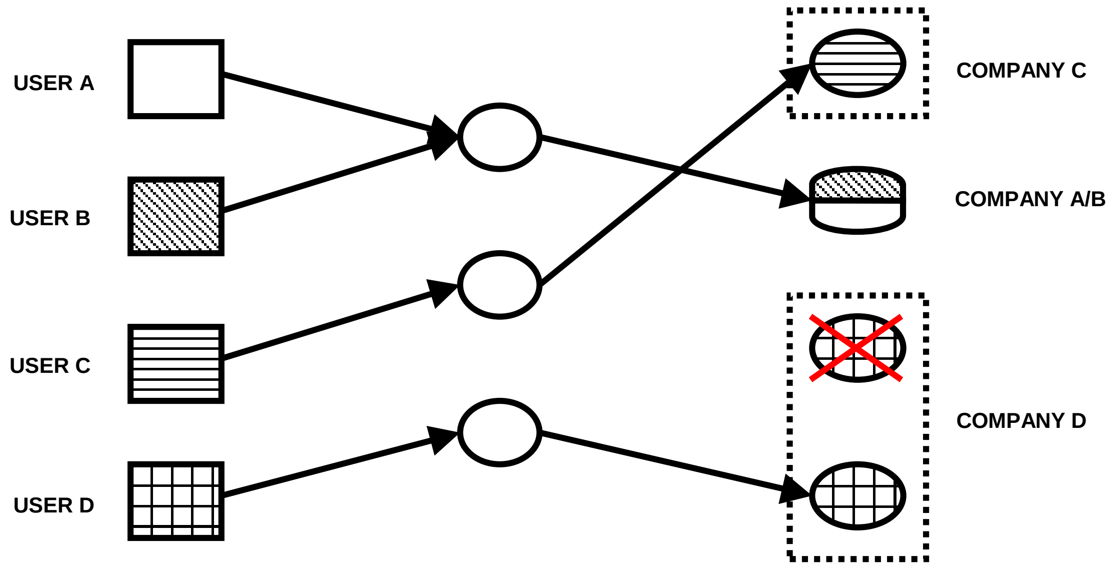
Details of the ASPizer toolkit services are as follows:

1. **Profiles** : This service deals with maintaining information unique to users, companies and applications. For users and companies, such information could include their preferences for various applications they access, details of the method of payment they use, etc. All users from a particular company share information stored in the profile for the company. Forms for entering and updating profile information are automatically generated and presented to the user (company admin for company profiles) when necessary. The ASPizer administration tool allows the ASP site administrator to review and update all available profiles if necessary. There is also a profile API available for use by the application programmer to provide users with a customized experience. Application profiles contain information necessary to create and maintain instances of the applications. These are created and maintained by the ASP site administrator using the ASPizer administration tool.

2. **Provisioning** : This service deals with creating and maintaining all required external resources for an application. These resources could be per application instance or per new user or company assigned to an application instance. Provisioning handlers can be registered with the service for various provisioning events. The service will automatically invoke the handlers when appropriate. Handlers can be standard (provided with ASPizer) or custom (created by application developer). The ASPizer administration tool can be used to specify handlers and associate them with provisioning events. Application developers can write custom handlers and package them with their applications.

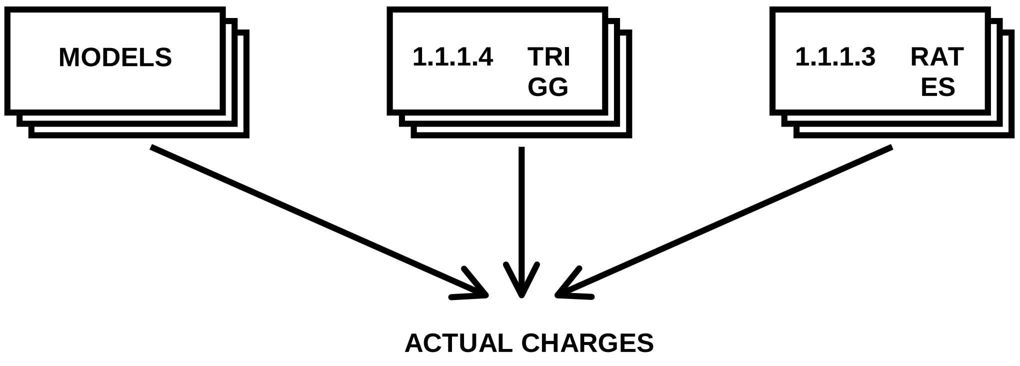
3. **Security** : This service deals with securing access to applications and application sub-functions. It provides a single-sign on to the ASPizer site. Security in ASPizer is role based – users are assigned roles, which are then allowed access to a set of resources. Security configuration is done using the ASPizer administration tool. Access to security configuration information is made available via a security API to the application programmer – this can be used to implement finer grain access controls.

4. **Service Level Agreements (SLAs)**: This service deals with the implementation of configurable service levels for different companies. Companies may elect whether or not they want to share a machine or an application instance. They may also select the level of reliability and availability they require. Based on these criteria, different application configurations will need to be used for users of a specific company. The picture below illustrates how users for four companies that have elected different service levels are handled. Company A and B have elected to share an application instance, Company C has elected to have an exclusive application instance and Company D has selected a highly available configuration with two application instances (one of which is shown as having failed). Based on these criteria, users from the four companies are routed to the appropriate application instances. SLAs are setup for companies using the ASPizer administration tool.



5. **Router** : The router service dynamically locates an application instance for a user based on a number of criteria including the specific request, the applicable security constraints, the SLA applicable for the user's company, the existence of failures in application instances, etc. There is no explicit configuration for the router service, its operation is implicitly dependent on the settings for other services such as the security service and SLA service.

6. **Billing** : This service deals with calculating the charges incurred by users and companies as they access and use the applications offered at a site. There are three aspects to billing – models, triggers and rates. Models deal with what metric is being used such as frequency of access, usage time or size of transactions. Triggers deal with which points are used for measuring the metrics. Rates deal with how much to charge for a certain measure of the metrics. Models and triggers typically depend on the type of application being considered while rates are typically set on a per company or user basis. The ASPizer administration tool can be used to setup billing models, triggers and rates.



7. **Licensing** : This service deals with enforcing different licensing policies for companies renting the applications at a site. License policies can include single user, multi user, node locked, etc. Licensing is specified per company per application it accesses and uses. The ASPizer administration tool can be used to setup licensing policies for companies.

8. **Payment collection** : The ASPizer toolkit supports the use of multiple payment collection methods through a customization layer. Based on the payment methods in use for a specific instance of the application, the toolkit automatically presents the payment choices to the user and collects all information relevant to the specific choice made by the user. The ASPizer administration tool allows for the setup and administration of payment methods.

9. **Audit** : This service helps track user activity across applications and services. This tracking can be useful for troubleshooting, non-repudiation and security investigations. Application developers can use the audit APIs to generate messages. All services automatically generate audit messages. The ASPizer administration tool can be used to view audit messages.

10. **Global session** : The global session service allows applications to share a user session. This is extremely useful for tight integration of applications – the shared session eliminates duplicate information entry and can be used to pass data between applications. Application developers can read and write session variables.

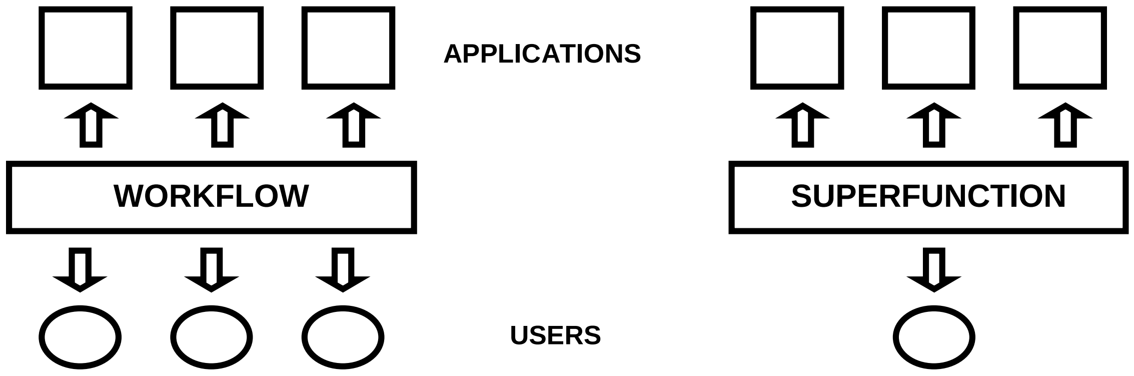
11. **Customizable look-and-feel** : This service allows for user and/or company level customization of the look-and-feel for an application. Customization is achieved by providing both the ability to select different looks for the elements of the application's UI as well as the ability to re-arrange these elements in different fashions on the screen. The ASPizer administration tool can be used to add new look-and-feel schemes and/or update existing look-and-feel schemes.

12. **Integrated Workflow** : This service allows users and companies to integrate applications with their business processes. Application developers need to provide special hooks into their applications to

enable integration with workflow. Users and companies can setup XML-based workflow templates to model their business processes using the ASPizer administration tool. They can also monitor the workflow instances active at any given time using this tool. ASP site administrators can use the ASPizer administration tool to setup master templates for standard business processes so that users and companies have a good starting point for creating their own templates.

13. **Super-functions** : This service allows the creation of wizards for specific complex tasks that span multiple applications. State can be carried from application to application via the global session. ASP site administrators can use the ASPizer administration tool to setup XML-based super-function templates.

The picture below illustrates how workflow and super-functions integrate users and applications. Workflow integrates multiple applications with multiple users while super-functions integrate multiple applications for a single user.



In the future, we plan on adding services such as a CRM service, a systems management integration service, a training service, a customer support service, etc.

3 Using the ASPizer toolkit

The first step to using the ASPizer toolkit is to construct an XML based deployment descriptor for use by the toolkit (ASPizer deployment descriptor). Note that this descriptor is in addition to the deployment descriptor typically required for the target application server (application deployment descriptor).

The ASPizer deployment descriptor contains configuration information for many of the ASPizer services mentioned earlier. The descriptor also contains application profile information that is used to create and manage instances of the application. We have provided below a sample ASPizer deployment descriptor for a simple payroll application.

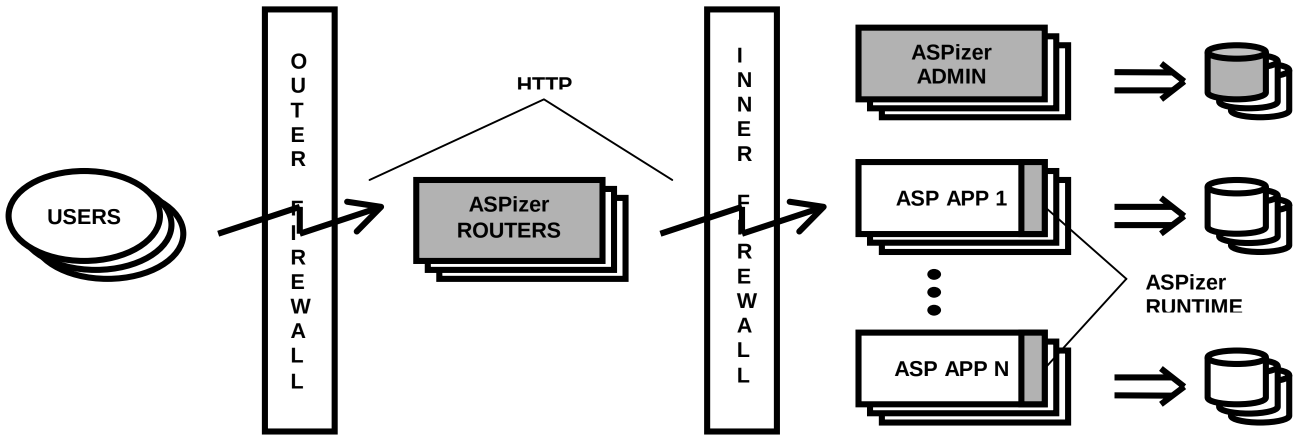
In this sample ASPizer deployment descriptor, there are sections describing the configuration for three services – Profiles, Provisioning and Billing. We have provided detailed descriptions for each section below:

- Profiles : The ASPizer toolkit provides for a set of standard pre-defined user and company profile attributes. Using the *extraProfileInfo* section, we are adding two additional company (*jobdescr, rate*) attributes and one additional user (*pymt_mode*) attribute. These attributes would represent application specific information that is pertinent for each company and user accessing it.

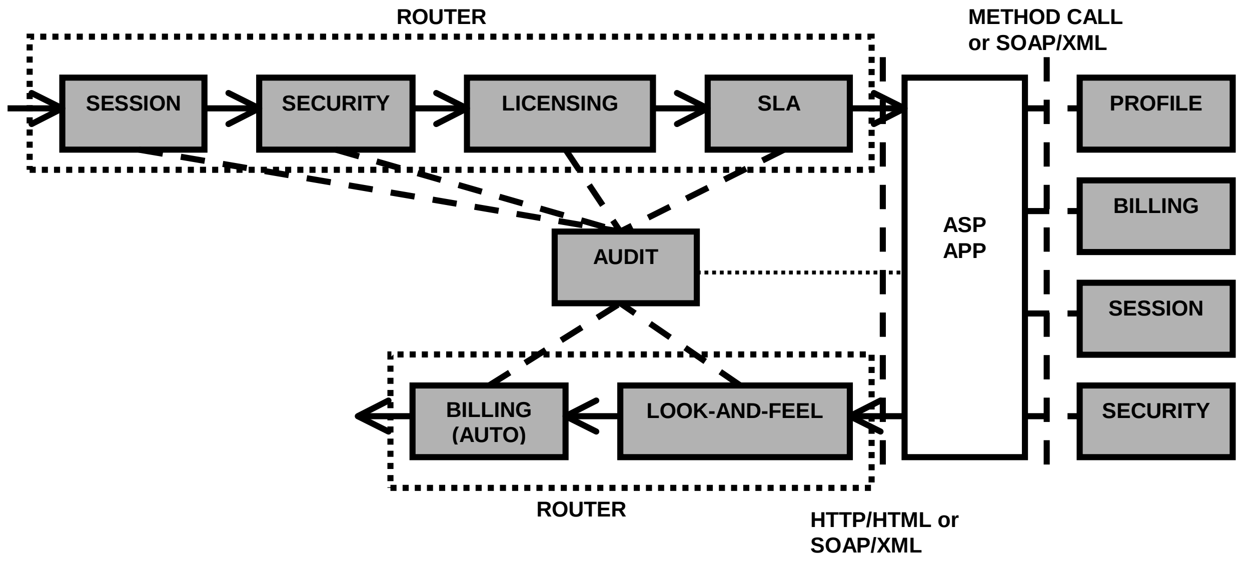
```
<aspFwkApp
    name="Payroll"
    locator="Payroll"
    description="Acme Payroll Application"
    version="1.0"
    vendor="Acme International"
>

    <extraProfileInfo>

       <companyAttr
           name="jobdescr"
           description=”Job Description”
           maxLength="128"
       />

       <companyAttr
           name="rate"
           description=”Rate of Pay”
           maxLength="64"
       />

       <userAttr
           name="pymt_mode"
           description=”Payment Method”
           maxLength="128"
       />

    </extraProfileInfo>

    <appInstance
       basePath="/Payroll"
       defaultPage="/index.html"
       host="ravenclaw.bizasp.com"
       port="80"
       createCompanyHandler="/CrtCompHlr"
       deleteCompanyHandler="/DelCompHlr"
    >

        <billableResource
           uri="/done.jsp"
           eventId="payroll_done"
           baseCharge="1000"
        />

    </appInstance>

</aspFwkApp>
```

- Provisioning : Using attributes of the *appInstance* section, we specify two custom provisioning handlers to be invoked during the addition (*CrtCompHlr*) and deletion (*DelCompHlr*) of a new company that will use the application.
- Billing : Using the *billableResource* section, we specify a charge of 1000 units to be added for every access of the URI “*/done.jsp*”. Note that we are using the frequency of access based billing model, with access to the specified URI acting as a trigger and a uniform rate of 1000 units for all companies accessing the application.

Once an ASPizer deployment descriptor is created, the ASPizer deployment tool is used to create and configure resources used by the ASPizer runtime. The ASPizer deployment tool also configures the target application server using the application deployment descriptor provided with the application. The ASPizer runtime can be installed in the target application server using a provided installation utility. Now the application is ready to be run in the ASP mode. The figure below illustrates the physical overview of an ASP site. All the shaded boxes represent ASPizer related applications and services.



The picture below illustrates a typical request/response flow in an ASPizer-based ASP site. Every request passes through the session, security, licensing and SLA service before reaching the application. The application can make requests to the profile, billing, session and security services via the corresponding APIs. The application's response is then passed through the look-and-feel and auto billing services before being sent back to the user. All the services and the application can write messages to the audit service anytime during this entire flow.



4 Implementation of the ASPizer toolkit

The ASPizer toolkit is completely implemented in Java and conforms to the J2EE standards. The toolkit has been designed to work with clusters of servers in a truly distributed fashion. This allows for a site that is highly scalable as well as fault-tolerant.

Most of the ASPizer runtime services are flexible and configurable and can be accessed directly via Java APIs. It is relatively easy to plug in custom implementations of a specific runtime service without affecting the rest of the runtime. All runtime services export data that provide information about their health and log enough information to help diagnose the causes of any problems that may occur. Site access services provided by the runtime include highly customizable Login, Logout and Router services. The Router controls all user access to applications enabling location transparency, security and license control. The "router pipeline" has well-defined components which can each be replaced by customized versions without affecting the other components, providing a very high degree of customizability and control to the site administrator. The Login and Logout applications are completely customizable to suit the needs of the site.

The ASPizer administration tool provides simple to use site administration and monitoring capabilities to the administrator. . The tool can be customized to accommodate site specific functions and look-and-feel. The ASPizer administration tool covers the following areas:

- Application management – this covers managing application instances, application security configuration, application profiles and application provisioning requirements among other things.
- User and company management – this covers managing user and/or company profiles, security roles for users, company security configuration, applicable billing rates for a company, company workflow templates and user and/or company look-and-feel schemes among other things.
- Aggregation management – this covers managing workflow master templates and super-function templates.
- Site-wide management – this covers managing licensing policies, billing models and triggers, payment methods, global session timeouts, nodes in the site and look-and-feel schemes among other things.

The monitoring capabilities of the ASPizer runtime services, combined with the administration tool help the site administrator check the health of the site and proactively address problems that may arise.

5 Summary

In this paper we have discussed the additional logic needed for an application to run in the ASP mode. As we have described, for an application to be leveraged through multiple market channels, for aggregations of applications to be offered in well-integrated offerings and for application offerings to be built in a collaborative manner by companies, a fairly extensive set of services are required. Our ASPizer toolkit provides this extensive set of services needed to truly meet the requirements of the ASP market and ensure success.