SAL:
A Service Level Agreement Language

Donna Eng Dillenberger
engd@us.ibm.com

September 22, 2003

Contents

An Introduction to SAL

The SAL language is designed to help Information Technology (IT) professionals and Business Process Modelers quickly build metrics, prototype, evaluate and optimize the configuration of their systems using Service Level Agreements (SLAs). Service Level Agreements are legal contracts used by business organizations to specify levels of performance, availability, storage, server and network capacity that an IT organization will provide to support the business processes. When these levels are not met, IT organizations are penalized by compensating the business in dollar amounts, for the time that lack of access to the computer infrastructure causes business transactions to be missed.

Examples of Service Level Agreements rules are:

- This IT organization will support the following 4 Business Processes:
    1. Trading Application
    2. Prospectus Browsing Application
    3. Risk Portfolio Analysis
    4. Maintenance, Background jobs
- The Trading Application:
    1. Will have a Mean Time to Recovery of 2 seconds (* where mean time to recovery implies that this application must never be inaccessible for longer than 2 seconds).
    2. All requests to the Trading Application will receive a response within 3 seconds.
    3. The profit from each Trading Application request is ten dollars.
    4. The IT cost of each Trading Application request should not exceed one dollar.
    5. If any of the above rules are missed, IT's payment to the business will be 100 dollars for:
        - Each transaction missed (because Mean Time to Recovery was not met). Where the amount of transactions missed will be an average of the transactions seen during the last business day interval before the computer infrastructure became inaccessible. The interval is determined by the amount of time the application is inaccessible.
        - Each transaction that failed to get a response within 3 seconds
- The Prospectus Browsing Application:
    1. Will have a Mean Time to Recovery of 5 minutes
    2. All requests to this Application will receive a response within 1 minute
    3. The profit from each request for this application is one dollar
    4. The IT cost of each request to this application should not exceed 50 cents
    5. No penalty to IT if the above rules are not met for Prospectus Browsers. The above rules are best effort.

- The Risk Portfolio Analysis Application:
    1. Will have a Mean Time to Recovery of 2 hours
    2. All requests to this Application will receive a response within 8 hours.
    3. The profit from each request for this application is two hundred dollars.
    4. The cost of each request to this application should not exceed10 dollars.
    5. If any of the above rules are missed, IT's payment to the business will be 100 dollars for:
        - Each transaction missed (because Mean Time to Recovery was not met
        - Each transaction that failed to get a response within 8 hours
- Maintenance and Background work:
    1. Will have a Mean Time to Recovery of 24 hours
    2. All requests to this Application will receive a response within 24 hours.
    3. The profit from each request for this application is one dollar.
    4. The cost of each request to this application should not exceed 50 cents.
    5. No penalty to IT if the above rules are not met for Prospectus Browsers. The above rules are best effort.

To specify the above using SAL, one would write:

```
Application Trading;
{       MeanTimeToRecovery = 2 seconds;
        ResponseTime = 3 seconds;
        Profit = 10 dollars;
        ITCost = 1 dollar;
        Penalty = 100 dollars;
}

Application ProspectusBrowsing;
{       MeanTimeToRecovery = 5 minutes;
        ResponseTime = 1 minute;
        Profit = 1 dollar;
        ITCost = 50 cents;
        Penalty =  0 dollars;
}

Application RiskPortfolioAnalysis;
{       MeanTimeToRecovery = 2 hours;
        ResponseTime = 8 hours;
        Profit = 200 dollars;
        ITCost = 10 dollar;
        Penalty = 100 dollars;
}
```

```
Application MaintenanceBackground;
{       MeanTimeToRecovery = 24 hours;
        ResponseTime = 24 hours;
        Profit = 1 dollar;
        ITCost = 50 cents;
        Penalty =  0 dollars;
}
```

The compiler for SAL will:

(1) Check that for each defined Application, there are MeanTimetoRecovery, ResponseTime, Profit, ITCost and Penalty values defined and specified in the right units.
(2) Transform the Application definitions above to java classes

The java code could then be fed into SLA deployment machines (these don't currently exist yet) that take the above definitions and:

(3) Creates Monitoring Agents for each Application and measures whether the above thresholds are violated
(4) In more sophisticated SLA deployment machines, forecast when the defined thresholds will be violated. Parameter passed into Forecast would request how far into the future a forecast is desired.
(5) Provisions/deprovisions/reconfigures IT routers, servers and storage to prevent Threshold violation from occurring.
(6) In more sophisticated SLA deployment machines, the IT adaptation would maximize profit, minimize cost, within the boundaries of not incurring IT penalties.

The programmer would invoke the above actions (3)-(6) by coding:

```
Actions;
{
        CreateAgents;
        Forecast (1 hour);
        Provision;
        Optimize;
}
```

For the above, the compiler would derive/calculate the right parameters to invoke the above fundtions:

- (3) CreateAgents
- (4) Forecast
- (5) Provision
- (6) Optimize

The implementations for 3-6 above are meant to be pluggable Different vendors can supply different implementations to differentiate themselves. The interfaces will be generated by the compiler but the package implementations will not be provided (these would be vendor specific).

Finally, the programmer would ask for Reports using the Report function:

```
Report;
{
        Filename;
}
```

This would print the results of Creating Agents, Forecast warnings, what provisioning, configuration actions were taken, and what optimization tradeoffs were invoked.

A key initial step in writing a SLA is to identify the Business Processes, the IT infrastructure will support. IT implementations should be driven by the business needs for the measurement of progress and fulfillment of business goals (e.g. ROI, EBIT). At a fundamental level, each organization has a general goal, e.g. maximize Return On Investment (ROI) or Earnings Before Interest and Taxes (EBIT). An organization's next step is to translate those goals into Lines of Businesses that can support maximizing these business goals. For example a Financial Institution may choose to offer three Lines of Businesses to maximize ROI:

- A Trading Service
- A Risk Portfolio Analysis Service
- A Prospectus Downloading Service

These three types of services all would have quarter profit and cost targets. The IT implementations built to support these services should help a Business measure how each service is progressing (number of transactions/day, IT cost of each transaction, profits of each service per day) to help the Business make decisions on:

- Whether to increase capacity for that service
- Whether to offer some services more frequently than other dependent on time of day, day of week or season of year
- Shut down that service (not profitable)
- Offer a new service.

The code for the above would look like:

```
BusinessGoal;
{
        Maximize ReturnOnInvestment;
        Minimize ITCosts;
}

ReturnOnInvestment;
{
        LineOfBusiness Trading;
        LineOfBusiness RiskPortfolioAnalysis;
        LineOfBusiness  ProspectusBrowsing;
}

ITCosts:
{
        Routers;
        Storage;
        Servers;
        Bandwidth;
        Power;
        Personnel = 20;
}
```

The compiler would take the above code and generate the java statements that would:

- Check that ReturnOnInvestment had associated Lines Of Business types
- Check that each LineOfBusiness type had associated Application structures defined (previous code sample that gives us IT metrics)
- Check that ITCosts had a structure delineating which physical resources were to be monitored, and that Personnel was assigned a value.
- Deploy monitoring agents (just calls to interfaces) to get NumberOfTransactions/day for each LineOfBusiness.
- Calculate ReturnOnInvestements = Sum of Profits from each LineOfBusiness transaction.
- Obtain ITCosts for each Line of Business transaction by deploying Monitoring agents on each ITCosts member (just a call to an interface)
- Use the above data from the Monitoring agents to calculate IT costs per Application
- Support the verbs FORECAST, PROVISION, OPTIMIZE by calling the correct methods and generating the data needed by each method (from the bullets above).

In the Service Level Agreement Language for this project, I propose to offer language types and operations that help Businesses:

- Translate Business Process Goals to Business Process Metrics
- Indicate to the IT infrastructure what IT measurements should be used to report on Business Process Metrics
- Indicate to the IT infrastructure what trade offs can be made when adding/deleting storage, network and server capacity to a SLA application based on cost and profit.

SAL is an intuitive, object-oriented, portable, powerful way to specify Business Goals. These Business Goals dynamically cause agents to be created to monitor the progress towards Business Goals and can call the right packages to optimize the IT infrastructure to support these goals

The foremost goal in the creation of this language was to make it easy to learn and straightforward to program. The user should be left to concentrate on the structure and design of their Business Processes, not the syntax of the modeling language. SAL was created to be consistent (strong type checking) and intuitive (type definitions are in the language of Business Architects), even to users who have limited programming experience.

By supporting the concept of objects, each Application of the Business has its own attributes. By breaking down the Line Of Business into its Application components, we believe that most users will find SAL to be conceptually intuitive to use.

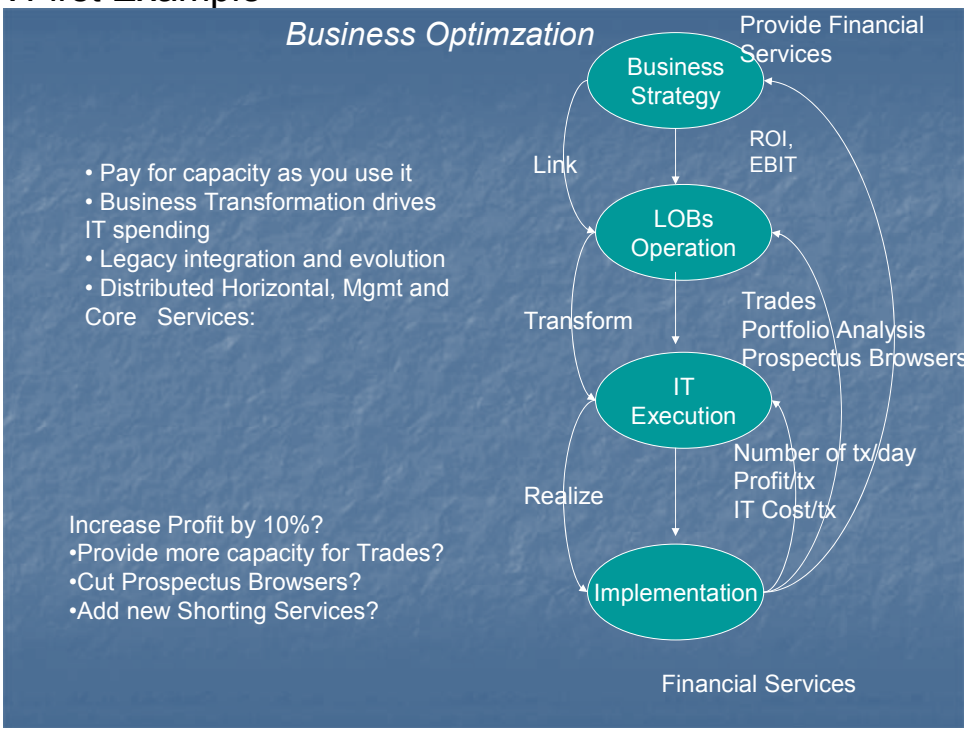
The compiler will accept a SAL specification as an input and generate Java source code. This source code can be integrated with a larger Java project and compiled with any Java compiler. Because Java is SAL's target platform, portability is limited only by the availability of a Java Virtual Machine.

With the power and simplicity of a robust SAL specification engine, the time to design, implement and test a Business Process' IT infrastructure requirements can be reduced by an order of magnitude. SAL's simple and intuitive language syntax ensures that most errors are detected at compile time and that compiled SAL code is as accurate as it can be. Monitoring Agents are automatically deployed. Profits are automatically calculated. IT Costs are automatically deduced. The deployment of SAL will focus its users on which Business Processes are the most profitable, at what time of the day, week, year, and which Business Processes are the most costly in terms of IT support to provide.

Tutorial

A Service Level Agreement specification is a sequence of definitions of Business Applications a corporation will purchase IT infrastructure to support. The IT infrastructure should measure the cost of Business Transactions, the profit generated from each Business Application and is able to signal a reconfiguration or automatically call reconfiguration packages to optimize profit and minimize cost.

A First Example



In the figure above, a Financial Institution decides that its business strategy is to provide premier Financial Services. The corporation measures how well it's providing these services based on Return On Investment and Earnings Before Interest and Taxes. The corporation decides it will create three Lines of Businesses to maximize ROI and EBIT. The Lines Of Businesses are: Trades, Portfolio Analyses and Prospectus Brwosers.

To measure how each Line of Business is doing, IT deploys these applications and measures (1) Number of Transactions (2) Profit for each transaction and (3) IT Cost of each transaction for each Line of Business Application: Trades, Portfolio Analysis and Prospectus Browsers.

The code that the Business Modeler would write was documented as examples in the above section. The code to deploy Agents to measure the IT execution metrics were automatically generated by the compiler and would be deployed by the execution of this code. SAL's compiler generates compiler errors if a Business Strategy does not provide Business Metrics to measure (e.g. ROI, EBIT), Lines of Businesses to calculate the Business Metrics and Members of an ITCost structure to automatically generate the code to calculate cost and profit for the Business.

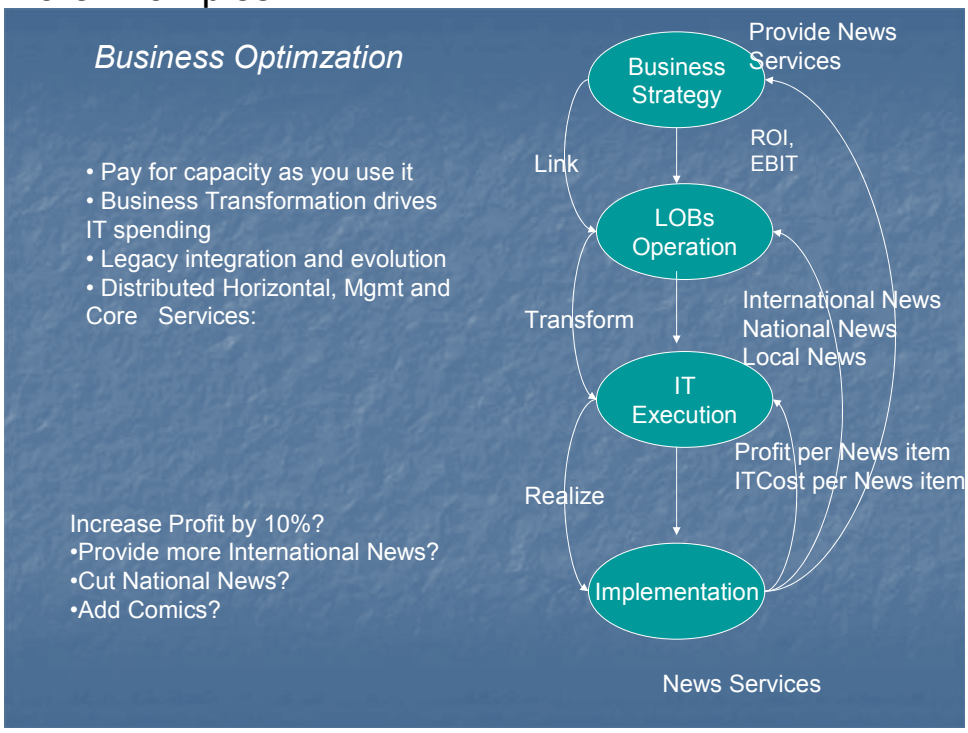
Compiling and Running SAL Spec Files

Once you have a simple example of a Business Process that you want to run through SAL, create a .sal file with your Application, Lines of Business and ITCost structures, compile it using SAL. If you have created a Business Process named ProfitableFinancialServices in a file called ProfitableFinancialServices.sal, you would compile it using:
$ java SALcompile ProfitableFinancialServices.sal

This will produce a file called ProfitableFinancialServices.java that contains java code implementing the monitors and metrics for this Business Process.

Compile the java file using javac and then run it using java:
$ javac ProfitableFinancialServices.java
$ java ProfitableFinancialServices

More Examples



In the figure above, a News Services Institution decides that its business strategy is to provide premier News Services. The corporation measures how well it's providing these services based on Return On Investment and Earnings Before Interest and Taxes. The corporation decides it will create three Lines of Businesses to maximize ROI and EBIT. The Lines Of Businesses are: International News, National News and Local News.

To measure how each Line of Business is doing, IT deploys these applications and measures (1) Number of News items (2) Profit for each news item (3) IT Cost of each news item for each Line of Business Application: International News, National News and Local News.

The code that the Business Modeler would write is:

```
BusinessGoal;
{
        Maximize ReturnOnInvestment;
        Minimize ITCosts;
}

ReturnOnInvestment;
{
        LineOfBusiness InternationalNews;
        LineOfBusiness NationalNews;
        LineOfBusiness  LocalNews;
}

ITCosts:
{
        Routers;
        Storage;
        Servers;
        Bandwidth;
        Power;
        Personnel = 200;
}

Application InternationalNews;
{       MeanTimeToRecovery = 2 minutes;
        ResponseTime = 5 seconds;
        Profit = 10 dollars;
        ITCost = 5 dollars;
        Penalty = 2 dollars;
}

Application NationalNews;
{       MeanTimeToRecovery = 5 minutes;
        ResponseTime = 1 minute;
        Profit = 1 dollar;
        ITCost = 50 cents;
        Penalty =  0 dollars;
}
Application LocalNews;
{       MeanTimeToRecovery = 1 minute;
        ResponseTime = 1 second;
        Profit = 200 dollars;
        ITCost = 10 dollar;
        Penalty = 100 dollars;
}

Actions;
{
        CreateAgents;
        Forecast (1 hour);
        Provision;
        Optimize;
}
```

```
Report;
{
        Filename;
}
```

The code to deploy Agents to measure the IT execution metrics would be automatically generated by the compiler and would be deployed by the execution of this code. SAL's compiler generates compiler errors if a Business Strategy does not provide Business Metrics to measure (e.g. ROI, EBIT), Lines of Businesses to calculate the Business Metrics and Members of an ITCost structure to automatically generate the code to calculate cost and profit for the Business and pass to forecasting, and optimization packages. If the verb PROVISION is requested, the compiler would also invoke libraries that would configure servers, storage and network to meet the APPLICATION requirements.

Project Plan

Project Timeline

The following deadlines were set for key project development goals.
10-08-2003 Language whitepaper, core language features defined
10-22-2003 Development environment and code conventions defined
11-07-2003 Language reference manual, grammar complete
11-14-2003 Parser complete
11-21-2003 Code generation complete
11-28-2003 Error recovery complete
12-02-2003 Code freeze, project feature complete

Software Development Environment

This project will be developed on Windows using Java SDK 1.4.1. The parser will be developed using CUP v0.10k, a Java variant of the yacc utility. The scanner will be developed using JFlex 1.3.5, a Java variant of the lex utility. The project will be tested using JUnit 3.8.1, a Java unit-testing framework.