SAL:
A Service Level Agreement Language

Donna Eng Dillenberger
engd@us.ibm.com

October 22, 2003

Document Changes

Date	Changes
9/22/03	Document Created
10/08/03	Included variables, control-flow, and functions.
10/22/03	Included Language Reference Manual

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. Between 9 am and 5pm will have a Mean Time to Recovery [1] of 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 for trades.
  4. The IT cost of each Trading Application request should not exceed one dollar.
  5. If the response time is greater than 3 seconds and it's after 9 am then a penalty of 100 dollars shall be incurred for each transaction.

- The Prospectus Browsing Application:
  1. Between 9am and 10 pm 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 two dollars
  4. The IT cost of each request to this application should not one dollar
  5. No penalty to IT if the above rules are not met for Prospectus Browsers. The above rules are best effort.

[1] Mean time to recovery entails that this application must not be inaccessible for longer than 2 seconds)

- The Risk Portfolio Analysis Application:
  1. Between 6pm and 12 pm, 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 exceed 10 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) between 6 pm and 12 pm.
     - Each transaction that failed to get a response within 8 hours

Maintenance and Background work:

1. Will have a Mean Time to Recovery of 4 hours between 1 am and 7 am
2. All requests to this Application will receive a response within 24 hours.
3. The profit from each request for this application is 3 dollars.
4. The cost of each request to this application should not exceed 2 dollars.
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;
{       For Scope = 900 to 1700;
        {
                MeanTimeToRecovery = 2 seconds;
         }
        ResponseTime = 3 seconds;
        Profit = 10 dollars;
        TxCost = 1 dollar;
        For Scope >= 900 && ResponseTime >=3 seconds;
        {
                Penalty = 100 dollars;
        }
}

Application ProspectusBrowsing;
{
        For Scope = 900 to 2200;
        {
                MeanTimeToRecovery = 5 minutes;
        }
        ResponseTime = 1 minute;
        Profit = 2 dollars;
        TxCost = 1 dollar;
```

```
        Penalty =  0 dollars;
}

Application RiskPortfolioAnalysis;
{       For Scope = 600 to 1200;
        {
                MeanTimeToRecovery = 2 hours;
        }
        ResponseTime = 8 hours;
        Profit = 200 dollars;
        TxCost = 10 dollars;
        For Scope = 600 1200 && (ResponseTime > 8 hours |
                                        MeanTimeToRecovery  >  2 hours);
        {
                Penalty = 100 dollars;
        }
}

Application MaintenanceBackground;
{       For Scope = 100 to 700;
        {
        MeanTimeToRecovery = 4 hours;
        }
        ResponseTime = 5 hours;
        Profit = 3 dollars;
        TxCost = 2 dollars;
        Penalty =  0 dollars;
}
```

The compiler for SAL will:

- Check that for each defined Application, there are MeanTimetoRecovery, ResponseTime, Profit, TxCost and Penalty values defined and specified in the right units.
- 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:

1. Creates Monitoring Agents for each Application and measures whether the above thresholds are violated
2. 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.
3. Provisions/deprovisions/reconfigures IT routers, servers and storage to prevent Threshold violation from occurring.
4. 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 (2)-(4) by coding:

```
Actions;
{
        Forecast;
        Optimize;
        Provsion;
}
```

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

- Forecast
- Optimize
- Provision

The implementations for 1-4 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).

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:

```
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 ReturnOnInvestments = 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.

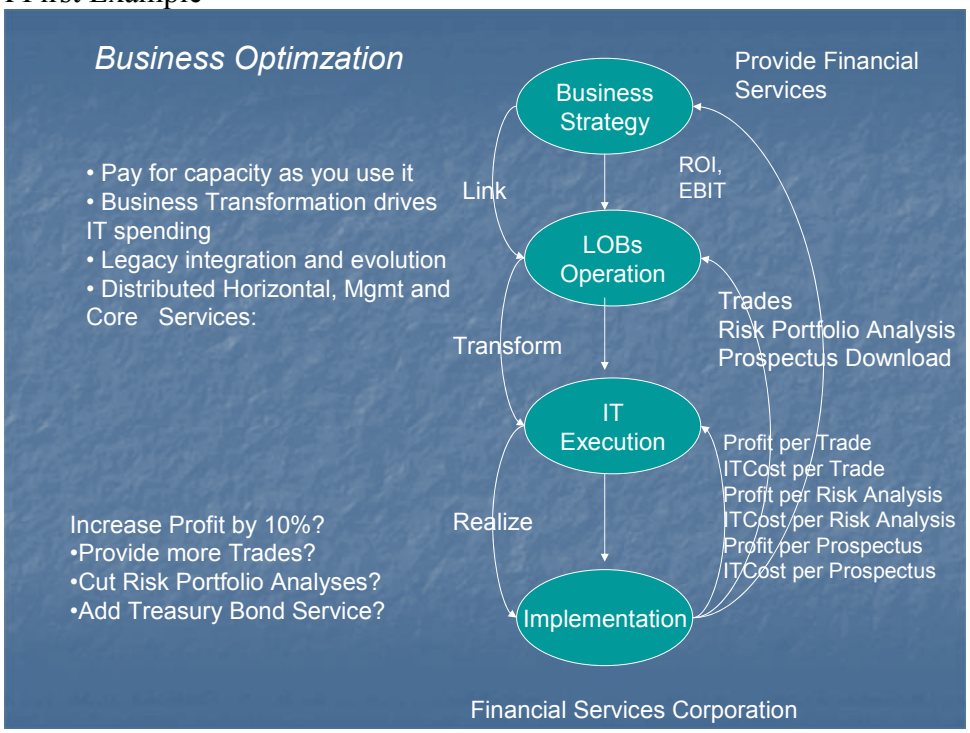
The compiler will accept a SAL specification as an input and generate Java source code. This source code can be integrated with a larger Java project and compiled with any Java compiler. Because Java is SAL's target platform, portability is limited only by the availability of a Java Virtual Machine.

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 Browsers.

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

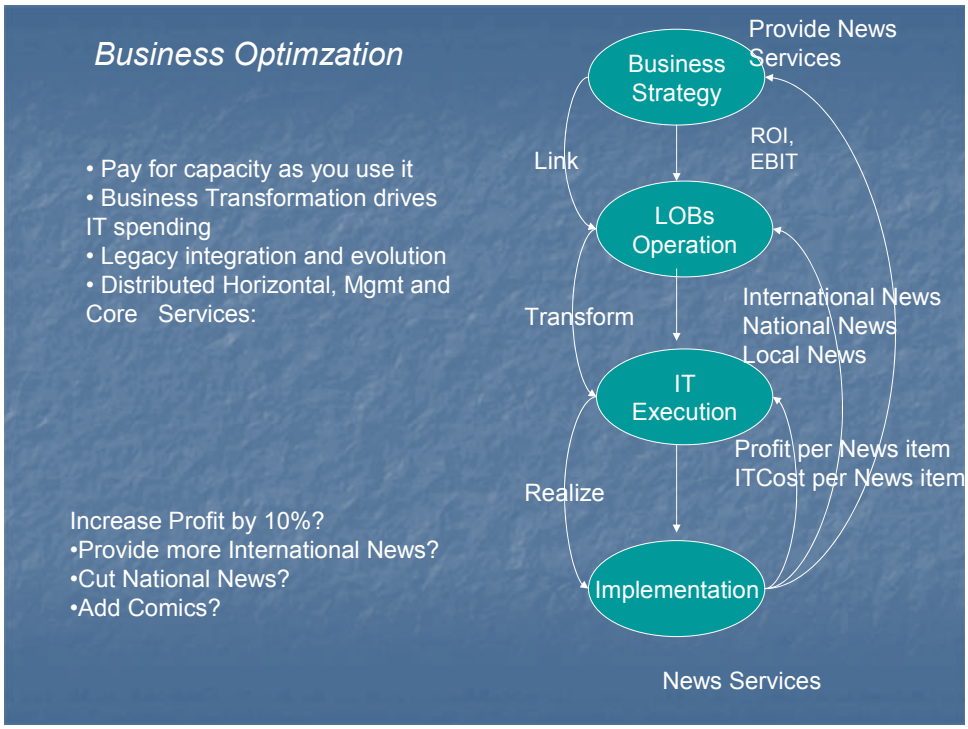
This will produce a file called ProfitableFinancialServices.java that contains java code implementing the monitors and metrics for this Business Process.

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:

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

```
Application InternationalNews;
{       For Scope = 1200 to 2400;
        {
                MeanTimeToRecovery = 2 minutes;
        }
        ResponseTime = 5 seconds;
        Profit = 10 dollars;
        TxCost = 5 dollars;
        Penalty = 2 dollars;
}

Application NationalNews;
{       For Scope = 800 to 1000;
        {
                MeanTimeToRecovery = 5 minutes;
        }
        ResponseTime = 1 minute;
        Profit = 6 dollars;
        TxCost = 2 dollars;
        Penalty =  0 dollars;
}
Application LocalNews;
{       MeanTimeToRecovery = 1 minute;
        ResponseTime = 1 second;
        Profit = 200 dollars;
        TxCost = 10 dollars;
        For Scope = 900 to 1700 && ResponseTime > 1 second;
        {
                Penalty = 100 dollars;
        }
}
ITCosts:
{
        Routers;
        Storage;
        Servers;
        Bandwidth;
        Power;
        Personnel = 200;
}
Actions;
{
        Forecast;
        Optimize;
        Provision;
}
```

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 TxCost 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.

Reference Manual

Grammar Notation

Grammar symbols are defined as they are introduced in this document. Regular expression notation has been used to make the productions more perspicuous. Symbols in italics are nonterminals. Quoted symbols are terminals. `s?' denotes the symbol s is optional. `s*' denotes the symbol s may be occur zero or more times. `s+' denotes the symbol s may occur one or more times. `(s|t)' denotes a choice between the symbol sequences s and t. Parentheses are also used to group symbols with respect to `?', `*' and `+'.

Lexical Conventions

A program consists of one Return On Investment declaration, an ITCost declaration, an Actions declaration and one or more Line of Business declarations defined in a *.SAL file. Programs are written using the Unicode character set. The precise version of Unicode used will be determined by the Java Virtual Machine used to run the SAL compiler. For more information, refer to your Java Developer's Kit documentation.

Line Terminators

The sequence of input characters is divided into lines by line terminators. Lines are terminated by the ASCII characters CR (\carriage return"), LF (\linefeed"), or CR LF. The CR LF combination is counted as one line terminator, not two.

Comments

Only C++-style comments are supported. A C++-style comment begins with the characters // and ends with a line terminator. /* and */ have no special meaning inside comments beginning with //

Whitespace

Whitespace is defined as the ASCII space, horizontal tab and form feed characters, as well as line terminators and comments.

Tokens

There are five classes of tokens: identifiers, keywords, operators, integers and separators. Whitespace is ignored except as a token separator. Whitespace is sometimes required to separate adjacent tokens that might otherwise be combined into one token (i.e., identifiers, keywords and constants). Token formation is greedy: the input is searched for the longest string of characters that could constitute a token.

Identifiers
An identifier is a sequence of letters or digits, the first of which must be a letter. There is no limit on the length of an identifier. Two identifiers are the same if they have the same Unicode character for every letter and digit. Identifiers that have the same external appearance may not by identical. For example, the Latin capital letter 'A' and the Greek capital letter `A' (\Alpha") are different Unicode characters that have the same appearance when displayed.

*Identifier -> letter (letter | digit)**

Keywords
The following identifiers are reserved for use as keywords, and may not be used otherwise:

ReturnOnInvestment	Forecast	dollars
LineOfBusiness	Provision	For
ITCost	Optimize	to
Scope	minutes	Routers
ResponseTime	seconds	Storage
MeanTimeToRecovery	hours	Servers
Profit	minute	Bandwidth
TxCost	second	Power
Penalty	hour	Personnel
Actions	dollar	

Integers
An integer is a sequence of digits. The limit on the length of an integer is 12 digits. Two integers are the same if they have the same Unicode character for every digit.

Integer -> (digit)+

Operators
The following symbol is an operator that is used to assign an integer to the following keywords Scope, |ResponseTime, MeanTimeToRecovery , Profit , TxCost , Penalty , Personnel:

=

Separators
The following ASCII characters are separators:

{ } ;

Return On Investment declaration
A “sal” file always starts with a Return On Investment declaration. This begins with the key word ReturnOnInvesment followed by a “;”. The next token is a “ {“ . The declaration ends with a “}”. Within the braces are one or more lines that begin with the key word “LineOfBusiness” followed by an Identifier for the Line of Businesses, followed by a “;”. For every line of business declared in the Return on Investment definition, a Line of Business Specification must follow.

Line Of Business declaration
A Line Of Business declaration begins with an Identifier of the Line of Business followed by a “;”. The next token is a “ {“ . The last token for a Line of Business Specification is a “}”. Within the braces are all of the following keywords followed by the operator “=” followed by an integer value for the key word, followed by a “;”. No keyword below can be missing. All of the keywords must have an integer assigned to it.

MeanTimeToRecovery – The duration of time the application must be available following an outage of the Line of Business Application. This is an integer followed by the keywords minute(s)|second(s)|hour(s).

ResponseTime – The time an end user must receive a response for a transaction initiated against the line of business. This is an integer followed by the keywords minute(s)|second(s)|hour(s).

Profit – The amount of US dollars one transaction for this Line of Business will bring to the company. This is an integer followed by the keyword dollar(s).

TxCost – The amount of US dollars one transaction cost in Information Technology (IT) services. This is an integer followed by the keyword dollar(s).
Penalty – The amount of US dollars missing the Mean Time To Recovery goal or the Response Time goal would incur on the IT organization. This is an integer followed by the keyword dollar(s).

Values for Mean Time To Recovery, Response Time and Penalty can be optionally specified to apply to only certain times of the day. This is done by using the “For” keyword followed by the “Scope” keyword, followed the operator “=”, followed by an Integer followed by the keyword “to” followed by an Integer, followed by a “;”. Valid values for Scope are 0 to 2400. The time format is hhmm, where hh = The hour time in US Military format, and mm = minutes. The beginning time must be less than the ending time. Scopes cannot span more than 24 hours. If no time of day scope is declared for Mean Time to Recovery, Response time or Penalty, the default scope is set to the full 24 hours in a day.

ITCosts Declaration
An ITCost declaration begins with the keyword "ITCosts" followed by a ";". The next token is a " {" . The last token for an ITCost specification is a "}". Within the braces are at least one of the following keywords followed by a ';"

Routers – Include costs of routers for the IT infrastructure supporting these Lines of Businesses.

Storage - Include costs of the storage (storage volumes, storage fabric, storage switches, storage appliances) for the IT infrastructure supporting these Lines of Businesses.

Servers - Include costs of the servers (Racks, Hardware, Operating Systems, Memory, Software) for the IT infrastructure supporting these Lines of Businesses.

Bandwidth - Include costs of the internet and intranet network bandwidth for the IT infrastructure supporting these Lines of Businesses.

Power - Include costs of the electrical power consumption required for the IT infrastructure supporting these Lines of Businesses.

Personnel – Include costs of the people required to run the IT infrastructure. This keyword is followed by the operator "=", followed by an integer, followed by a ";". The integer denotes how many people are required to run the IT infrastructure.

All of the costs above will be monitored and aggregated to a day's cost (24 hours).

Actions Declaration
An Action declaration begins with the keyword "Actions" followed by a ";". The next tokwn is a " {". The last token of the ITCosts declaration is a "}". Within the braces are at least one of the following keywords followed by a ";"

Forecast – This action will forecast the next day's arrival rate, revenue, profit, cost and penalty.

Optimize – This action will calculate which Line Of Business would bring the most profit if given additional IT resources. This action cannot be issued without first issuing "Forecast".

Provision – This action will deploy new IT resources (servers, storage, network) needed to instantiate a new instance of the Optimal Line of Business supporting applications. This action cannot be issued without first issuing "Optimize".

Service Level Agreement Specifications

A Service level Agreement specification is a fille that contains a Return On Investment declaration followed by any number of Line of Business declarations, followed by an ITCost and Action declaration. A specification is compiled into a Java class. Executing the generated class will begin execution of the Service Level Agreement.

SLASpecification -> ReturnOnInvestment declaration
(LineOfBusiness declaration)+
ITCost declaration
Actions declaration

Names

A name is bound by a declaration and is available at any point in the specification that follows the declaration. A name must be unique within the specification.

Return On Investment Declaration

A specification must begin with a Return On Investment declaration. There may be only one Return On Investment declaration in a specification. A Return On Investment declaration takes the form:

ReturnOnInvestment Declaration -> 'ReturnOnInvestment' ';'
'{'
('LineOfBusiness' Identifier)+
'}'

Line of Business Declaration

One or more Line of Business declarations must follow a Return On Investment declaration. A Line of Business declaration takes the form:

Line of Business declaration -> Identifier ';'
('For Scope' = Integer 'to' Integer ';)?
'MeanTimeToRecovery' = Integer ';'|
'Responsetime' = Integer 'second' 's?' ';'
';'?
'Profit' = Integer 'dollar' 's?' ';'
'TxCost' = Integer 'dollar' 's?' ';'
'Penalty' = Integer 'dollar' 's?' ';'
'}'

IT Costs
Following the Line of Business declarations, is the IT Costs declaration. There may be only one IT Costs declaration in a specification. An IT Costs declaration takes the form:

IT Costs Declaration -> *'ITCosts' ';'*
'{'
'Servers;'
'Routers;' ?
'Storage;' ?
'Bandwidth;' ?
'Power;' ?
'Personnel' (= Integer)? ';'
'}'

Action
Following the IT Costs declaration, is the Action declaration. There may be only one IT Costs declaration in a specification. An Action declaration takes the form:

Action Declaration -> *'Action' ';'*
'{'
'Forecast' ';'
('Optimize;' ';') ?
('Provision;' ';') ?
'}'

Appendix A

SAL Grammar
The following lists all of the grammar productions described in the Language Reference section.

SLASpecification -> ReturnOnInvestment declaration
(LineOfBusiness declaration)+
ITCost declaration
Actions declaration

ReturnOnInvestment Declaration -> 'ReturnOnInvestment' ';'
'{'
('LineOfBusiness' Identifier)+
'}'

Line of Business declaration -> Identifier ';'
('For Scope' = Integer 'to' Integer ';)?
'MeanTimeToRecovery' = Integer ';'|
'Responsetime' = Integer 'second' 's?' ';'
';'?
'Profit' = Integer 'dollar' 's?' ';'
'TxCost' = Integer 'dollar' 's?' ';'
'Penalty' = Integer 'dollar' 's?' ';'
'}'

IT Costs Declaration -> 'ITCosts' ';'
'{'
'Servers;'
'Routers;' ?
'Storage;' ?
'Bandwidth;' ?
'Power;' ?
'Personnel' (= Integer)? ';'
'}'

Action Declaration -> 'Action' ';'
'{'
'Forecast' ';'
('Optimize;' ';') ?
('Provision;' ';') ?
'}'

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.

Bibliography

[1] Conway, Christopher, Li Cheng-Hong, Pengelly Megan. Pencil: A Petri Net Specification Language for Java. Columbia University, COMS W4115, December 2002