

CSEsolutions  
Technical White Paper on the  
RoboSuite by Kapow Technologies

## Introduction

### The Problem and the Business Case

The Web is a wealth of information. However, the Web's use is limited due to its size, dynamic nature, and unstructured format. Even if you're lucky enough to find the information you need, how do you get updated when that information changes? And how do you process that information to suit *your* needs? For example, how would you create an intra-website showing today's Web top stories related to your company? You could hire a person to surf and cut'n'paste the Web all day, but wouldn't it be much more efficient if the information you want can be automatically delivered directly to you, on-demand, on-schedule and in an easy-to-process format, such as XML?

The Web is also an opportunity for new ways of doing business. However, the Internet is meant for browsing and not to be used as an integration platform for business systems. For example, how would you create a web service that gives its users an aggregated view of all their bank accounts? You could try to convince the banks to allow you to snoop around inside their bank systems (good luck!), but wouldn't it be much simpler for everybody if your system could interact directly with the banks' existing websites that already provide account-viewing functionality?

### The Solution: RoboSuite

The solution is to create well defined and easily manipulated interfaces to websites and web services that allow you (or your business systems) to interact with websites with the same ease you interact with relational databases.

RoboSuite is a suite of applications that allow you to create, debug, manage, and interact with those interfaces, called robots. With RoboSuite you can easily create robots that extract relevant news stories from a selection of websites, or that seamlessly interact with the web services of several banks.

If you know what kind of web information you're are looking for and on what websites that information appears, then RoboSuite can help you retrieve that information whenever you want, and in the format that you want, such as in database format or XML format.

If you want to offer your customers a service based on third party web services, then RoboSuite can help you interact with those web services in a robust and cost-efficient way.

### The Solution Provider: CSEsolutions

CSEsolutions is a reseller of the RoboSuite is developed by Kapow Technologies, a company dedicated to deliver world-class robot technology that allows you to easily and automatically interact with websites regardless of their content or structure.

CSEsolutions in addition to selling the RoboSuite tool set, also provides RoboSuite training, first level support and technical services. The technical services include robot building and system architecture analysis in order to maximize the use of the RoboSuite.

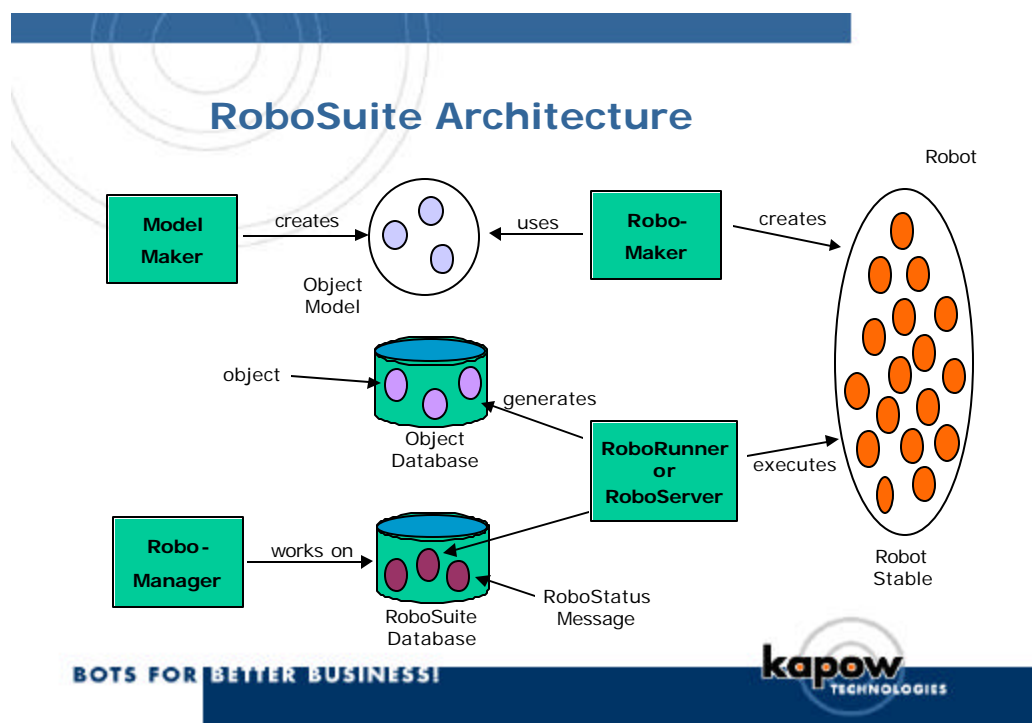
## RoboSuite

### Overview

RoboSuite is a suite of applications whose common purpose is the creation, execution, maintenance, and management of robots. RoboSuite consists of five applications: ModelMaker, RoboMaker, RoboRunner, RoboServer, and RoboManager.

Briefly, ModelMaker creates the *object models* for the objects that the robots use when interacting with a website, e.g. an article object model or an account balance object model. RoboMaker creates and debugs the robots, and RoboRunner and RoboServer executes the robots in a robot farm. RoboManager is used to manage the robot life cycle, including monitoring the status of robots.

The RoboSuite Architecture diagram below illustrates how the RoboSuite applications work together. Note, however, that the applications can work independently of one another. For example, you can use RoboRunner to run robots created and maintained by another company. The terms used in the diagram will be explained later.



### What Is A Robot?

"Robot" is the single most important concept in RoboSuite so it makes sense defining exactly what a robot is. From a low-level point of view, a robot is file with the file extension ".robot". The content of the file is an XML representation of the robot. The typical file size of a robot is 20-80 KB.

From a user point of view, a robot is a means to accomplish some specific task, such as extracting all articles from the cnn.com website, or submitting an order at an online bookstore. The robot is created in RoboMaker by visually connecting *steps* containing highly configurable building blocks called *tag processors*.

From a programmer point of view, a robot is a program written in a special language with its own syntax and semantics. The syntax is made up of steps, containing tag processor constructs, which can be linked together using *edges*. The semantics, or meaning, of the program determines what happens when the robot is run. As such, RoboMaker is a visual programming environment complete with testing and debugging facilities.

From an architectural point of view, a robot is a means of interfacing to remote websites allowing advanced interaction with the websites, as they were structured databases.

From a business point of view, a robot means new opportunities for doing business, both when it comes to reducing costs (by doing things in new ways) and exploiting new markets.

### RoboSuite Applications

RoboSuite consists of the following five applications: ModelMaker, RoboMaker, RoboRunner, RoboServer, and RoboManager. They are briefly described below.

#### ModelMaker

In ModelMaker, you create a *domain model*, or description, of the objects that the robots use when interacting with websites. A domain model contains an *object model* for each object type, and an *object* is an instance of such an object model in much the same way that objects are instances of classes in an object-oriented programming language. Unlike a class, however, an object model does not have any methods associated with it; it only contains *attribute models*. In academic terms, the object model would likely be called an *entity model* and the objects would be called *entities*. (Note that, in order to minimize user-exposed terminology, there is no distinction between models and instances in the RoboSuite GUIs.)

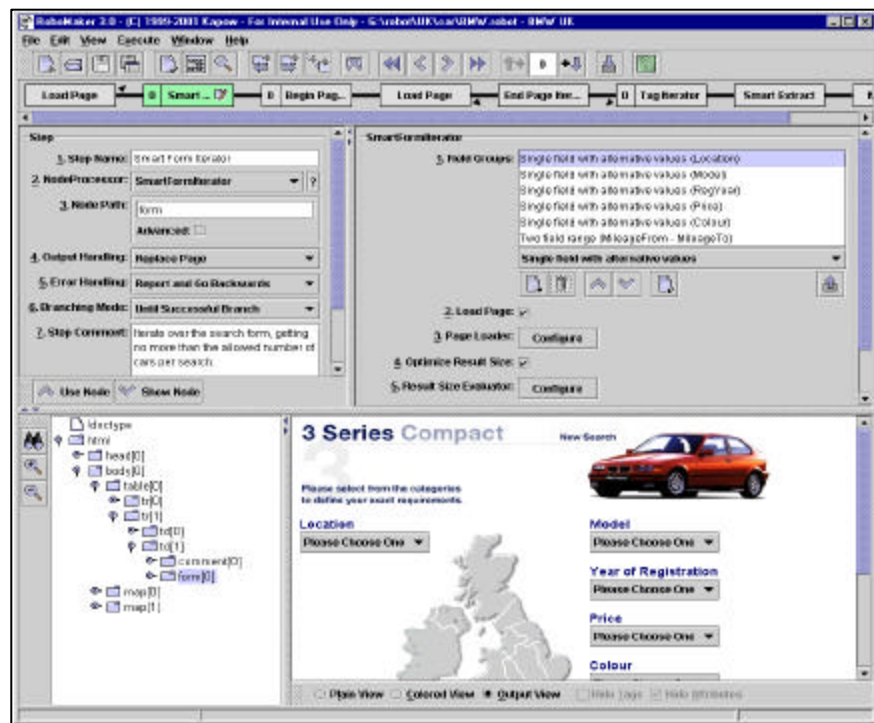
For example, in order to create a robot that extracts news from a website, we need to create an object model for news objects. Such a news object model would contain attribute models like title, body, author, URL, creation date, etc. Instances of the news object model (i.e. the news objects) would be the news themselves with attribute values such as "Robots roam the Net" (title), "Robots crawl the Net tirelessly in search of..." (body), "John Doe" (author), etc.

In ModelMaker, you group one or more object models into a domain model. The domain model can be saved as a ".model" file (in XML format) and be imported directly into RoboMaker, where robots, utilizing the domain model, can be created.

For example, once you have created a news object model, and potentially other object models as well, you group them into a domain model. This is then imported into RoboMaker and you can then start writing robots that extract news stories from the Web.

#### RoboMaker

In RoboMaker, you create the robots, one at a time. Before writing a robot, you decide on the task(s) that you want the robot to perform. An example task could be to extract some user's stock portfolios from a website. Before writing the robot, you need one or more domain models containing object models describing the objects you need to interact with the website; in this case you need an object model for the stock portfolio and an object model for the user information, containing attribute models such as username and password. A user information object can be provided to the robot as part of its query parameters when the robot is executed. In that way, the same robot can extract stock



portfolios for different users.

Once you have decided on the task, you create the robot that implements this task. You do this in much the same way that you would realize the task in a browser, namely by pointing and clicking.

A web page, such as an HTML or XML page, consists of tags and text. You build the robot program by manipulating the tags using a pre-defined set of highly configurable tag processors. For example, there is a tag processor for loading a web page, another for submitting an HTML form, a third for iterating over some tags, etc. Also, there are tag processors for extracting information from a tag and storing it in an object.

From a technical point of view, the entire robot is a graph, where each node, called a *step*, contains a name, a *tag finder*, a tag processor, and an error handling mechanism. The nodes are connected via *edges*. There are different kinds of edges that affect the runtime behavior of the robot.

RoboMaker contains a sub-application, RoboDebugger, for testing and debugging robots. In RoboDebugger, you can see the extracted objects, and the errors generated, if any. You can also set breakpoints to control the robot execution.

### RoboRunner

RoboRunner executes robots. RoboRunner has no graphical user interface (GUI) but is, instead, invoked directly from the command-line. This means that it works in conjunction with third-party schedulers, such as Unix's Crontab program. In effect, RoboRunner offers scheduled (batch) execution of robots.

In understanding the full flexibility of RoboRunner, the concept of *environment* is important. Each robot is run in an environment. This environment contains sub-environments for storing extracted objects, sending status messages and handling administrative information about the robot and its execution. There are many different implementations of these sub-environments allowing extracted objects to be stored in a relational database, or in a file in CSV or XML format; status messages can be stored in a database, in a file, or sent as an email; and administrative information can be read from a database or be provided as part of the environment. The exact configuration of the environment is determined at runtime, i.e. at the invocation of RoboRunner.

RoboRunner can run robots concurrently, i.e. multiple robots at a time. The robots are loaded as they are run, from locations specified by URLs.

### RoboServer

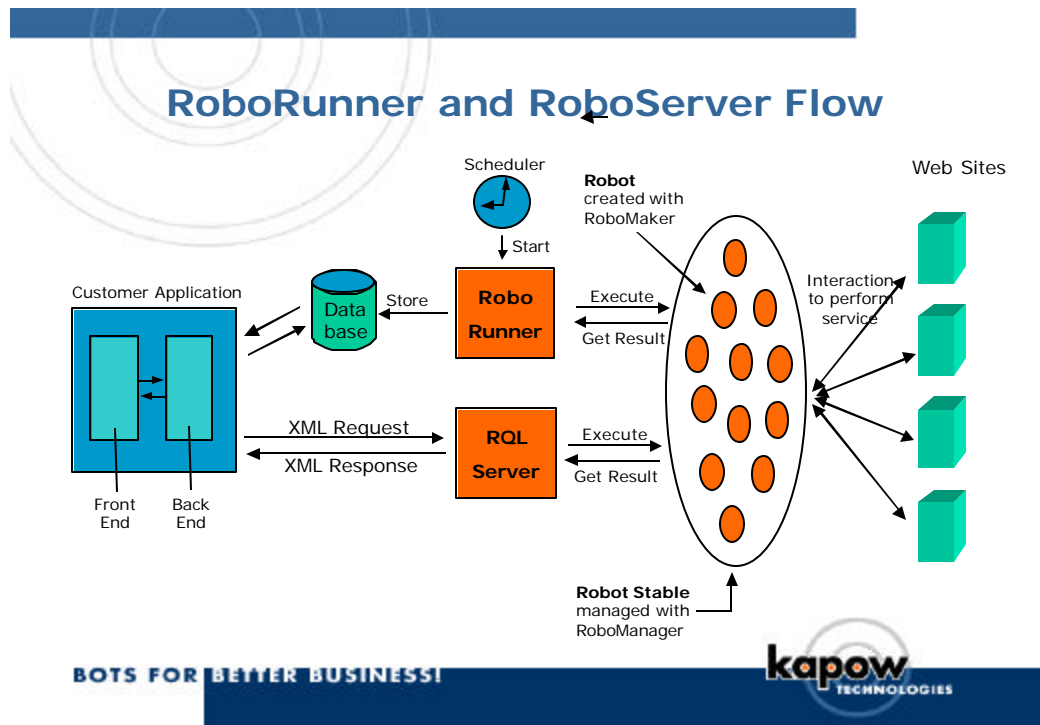
RoboServer also executes robots. However, unlike RoboRunner, RoboServer is a server whose service is to execute robots. Clients then trigger the execution of robots.

The interface of RoboServer is XML according to a DTD named RQL (Robot Query Language). Clients send RQL requests to the RoboServer and it replies with RQL responses.

RoboServer contains a sub-application named RoboClient for sending and receiving RQL constructs and viewing the transmitted XML. However, a typical client will be part of a customer back-end and act as the interface to a robot farm.

With RoboServer, programmatic interaction with websites and web services becomes easy. By sending an "Execute Robot RQL Request" to RoboServer, the client will receive RQL responses containing the extracted objects and robot-generated status messages. The syntax of the "Execute Robot RQL Request" allows you to configure the robot execution environment and hence potentially store the extracted objects in a database instead.

The diagram below summarizes how a typical customer application interacts with remote websites and web services via RoboRunner and RoboServer.



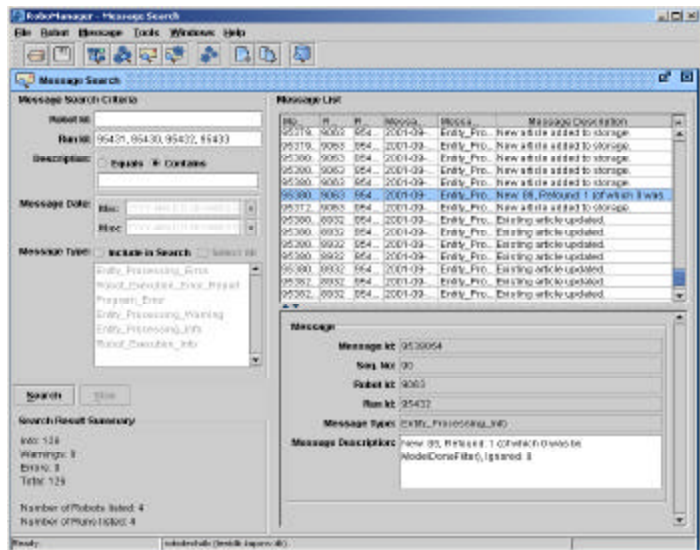
## RoboManager

RoboManager manages the *robot life cycle*. In its life, a robot cycles through the phases of birth, test, production, and death. To stay in the production phase, a robot needs to be monitored and kept in good health.

From a technical point of view, RoboManager is best thought of as a database client working against a RoboSuite database. The RoboSuite database contains tables for Robots, RobotRuns, Messages, etc. and

RoboManager allows you to quickly locate specific robots and robot runs, including the status messages generated during runs, such as error messages stating that a robot could not run due to drastic website changes and hence needs to be fixed.

In RoboManager, you can group robots, e.g. English news robots. RoboManager also contains a simple GUI for running a robot.



## Technology and Requirements

RoboSuite is a 100% pure Java application. Therefore, it will run on any platform for which there exists a Java Runtime Environment (JRE) v1.3 or later. Production-ready JREs exist for all major platforms including Windows, Unix and Linux.

RoboSuite is designed according to a component-based architecture (JavaBeans) offering a clear and highly robust and scalable design, where new components can be easily replaced and added, such as the addition of new (potentially customer-specific) components, such as new RoboMaker tag processors and new RoboRunner/RoboServer environments.

RoboSuite scales well. Experience shows that a single person can build and maintain about a thousand average-task robots, and a single PC-like machine dedicated to executing robots can execute about a thousand average-task robots a day. More people and machine power can be added on-demand with no loss of productivity.

Experience shows that, once setup, RoboSuite is easy to use and maintain. The user-intensive applications, ModelMaker, RoboMaker and RoboManager, are intuitive and require only basic knowledge of HTML and some knowledge of patterns (regular expressions). RoboRunner, RoboServer, and tasks related to database management might require the attention of a systems administrator.

The object database, wherein extracted objects can be stored, can be any database that is SQL-compatible and for which there exists a JDBC-driver. (A JDBC-driver exists for all major databases including Microsoft SQL Server, DB2, Sybase, Oracle, and MySQL.) The RoboSuite database, wherein robot-generated status messages can be stored, must be a MySQL, Sybase, or Microsoft SQL Server database.

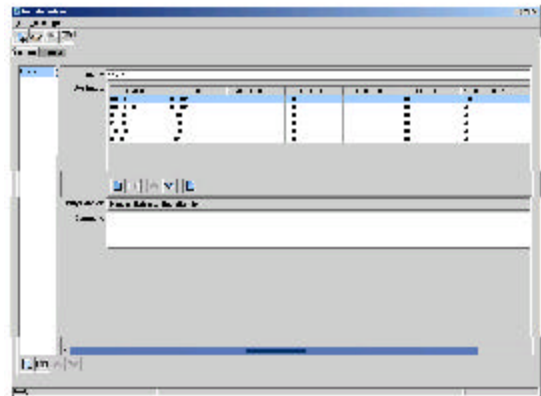
## ModelMaker

### Overview

ModelMaker is the RoboSuite application responsible for creating object models used by robots when interacting with a website. An object model, e.g. an article object model, consists of attribute models, e.g. title, body, author, and URL.

In ModelMaker, you create object models in much the same way you create tables in a visual database management environment. Once created, you can import the object models in RoboMaker and start writing robots that utilize them, e.g. article extraction robots.

As a convenience feature, ModelMaker can also generate the SQL queries needed to create database tables matching the object models. This allows you to store the objects your robots extract from the Web in your own SQL-compatible database.

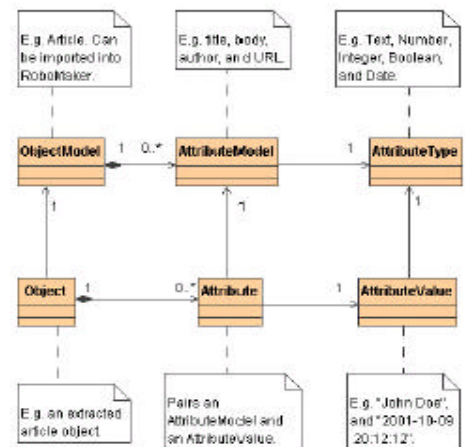


### Architecture

ModelMaker utilizes the JavaBeans component model. Each concept is modelled as a JavaBean. The diagram to the right shows a conceptual class diagram for ModelMaker.

The diagram reads as follows: An *object model* consists of *attribute models*, each having an *attribute type*. An *object* is an instance of some *object model* and consists of *attributes*, each pairing an attribute model and an *attribute value*.

Standard templates for object models and attribute models can be plugged in. New attribute types can also be plugged in. Attached to each object model (not shown in the diagram) is an *object handler* that determines how an extracted object is handled (i.e. preprocessed and stored). *Object handlers* can also be plugged in.



## Requirements

From a technical perspective, you need a computer with JRE v1.3 or later. ModelMaker is neither memory nor CPU intensive, so any Pentium-like machine will do. ModelMaker takes up less than 20 MB of hard disk space.

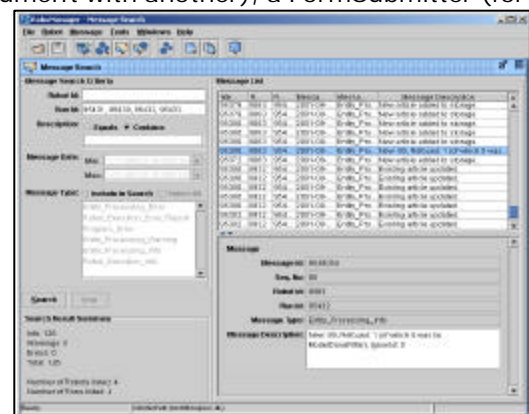
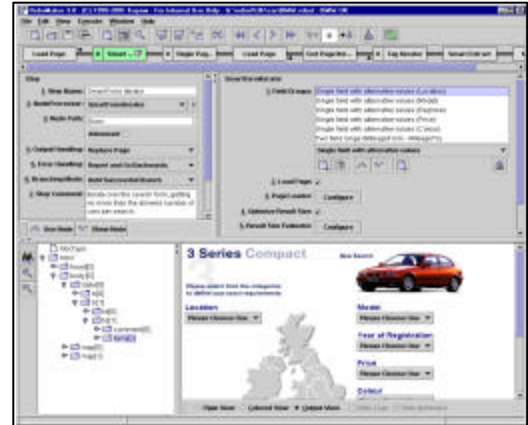
From a human resource perspective, you need a person capable of researching and putting together a model describing some object. This is not hard and can be learned in 1/2 hour.

## RoboMaker

### Overview

RoboMaker is the RoboSuite application responsible for creating, debugging, and fixing robots. A robot is a program, and RoboMaker is a visual programming environment for programming robots. The robot program is made up of *steps* containing building blocks called *tag processors*. Steps are connected via *edges* that control the program flow.

In many ways, RoboMaker is like a state machine. The state consists of the current document (such as a web page) and a set of objects (such as an article being extracted). The transitions between states are called steps and consist of a tag processor, a tag finder, and an error handling mechanism. The document is shown both using a DOM (Document Object Model) and a browser window. The tag processor processes a tag, identified by the tag finder, in some way and may, as a result, change the state. More than 40 standard tag processors are available, including a PageLoader (for replacing the currently loaded document with another), a FormSubmitter (for submitting an HTML form), a TagIterator (for iterating over a set of tags), and a SimpleExtractor (for extracting a text string from the document into an attribute of one of the objects). The extractors can optionally transform the extracted text string by using a list of *text processors*. More than 20 standard text processors are available, including text processors for uppercasing, date formatting, etc. RoboMaker also includes RoboDebugger for debugging robots. In RoboDebugger, you can run the robot program from a given location, set breakpoints to control the execution, view the current state and view extracted objects and generated messages.



### Architecture

RoboMaker (and RoboDebugger) is designed according to the Model-View-Controller design pattern. The Model is made up of the robot program, the state, and various other pieces of control information. The View is the GUI as seen by the user. The Controllers are the actions that the user performs. The Model is guaranteed to be consistent at all times and cause Controller actions to fail if they change the model in an invalid manner.

RoboMaker utilizes the Java Beans component model. Object models (made in ModelMaker), tag processors, and text processors can be added as plugins.

RoboMaker's execution layer was based on a state machine for several reasons: (1) It increases robustness because the state can be guaranteed not to become inconsistent, (2) since the state follows the execution thread it allows the same robot instance to be run in parallel using multi-threading, (3)

step output states can be cached and reused resulting in a huge performance increase, and (4) the state can be persisted and loaded at a later time.

### What Robots Can and Cannot Do

Basically, a robot can do whatever you can do in a browser. However, robots are limited as to what they can do by the tag processors (and text processors) available. New tag processors are continuously being added and improved and hence the capabilities of robots continue to increase. Customers can also write their own tag processors.

The current collection of tag processors is tuned for interacting with web pages, such as loading pages, submitting forms, and extracting content. Needless to say, there are not tag processors for everything. For example, there is currently no tag processor for executing JavaScript. (However, the lack of a JavaScript tag processor has never really been a problem, since the user can work-around most JavaScript quite easily by using other tag processors.)

### Robustness

As a result of website changes, a robot may fail and need to be fixed. A robust robot is a robot that can withstand non-dramatic website changes. In RoboMaker, the key to writing robust robots lies in the *tag finder*. Tag finder is the mechanism used to locate the tag on which the tag processor should work. (This is similar to XPath in XSLT.) If a web page changes, then the tag finder might not be able to find the tag to pass to the tag processor, causing the robot to fail. The tag finder can be programmed using either absolute indexing or relative indexing. In absolute indexing, you specify the exact location of a tag from the root node of the DOM. In relative indexing, you specify its location relative to some other node that can itself be relative to another node, and so on). Robustness increases as the indexing becomes more relative. However, risk also increases because relative indexing might accidentally index the wrong tag.

Typically, the trade-off between robustness and risk is determined by the tasks the robots perform and whether risk is acceptable. For example, if you extract news stories from a huge amount of websites, then you probably want to minimize robot maintenance by making them as robust as possible. However, if you plan to build a robot-driven market place on the Internet, then you probably want your robots to fail if the websites, through which the orders are submitted, change to avoid the risk of accidentally placing orders wrongly.

### Requirements

From a technical perspective, you need a computer with JRE v1.3 or later. RoboMaker requires 256 MB RAM and a Pentium-like machine (300+ Mhz). RoboMaker takes up less than 20 MB of hard disk space.

From a human resource perspective, you need a person understanding HTML and patterns (regular expressions), at least at some basic level. Experience shows that people can write a robot (that does something useful!) on their first day of learning RoboMaker.

If you would like to try a demo version of RoboMaker, please contact [RoboMakerDemo@csesolutions.com](mailto:RoboMakerDemo@csesolutions.com).

## RoboRunner

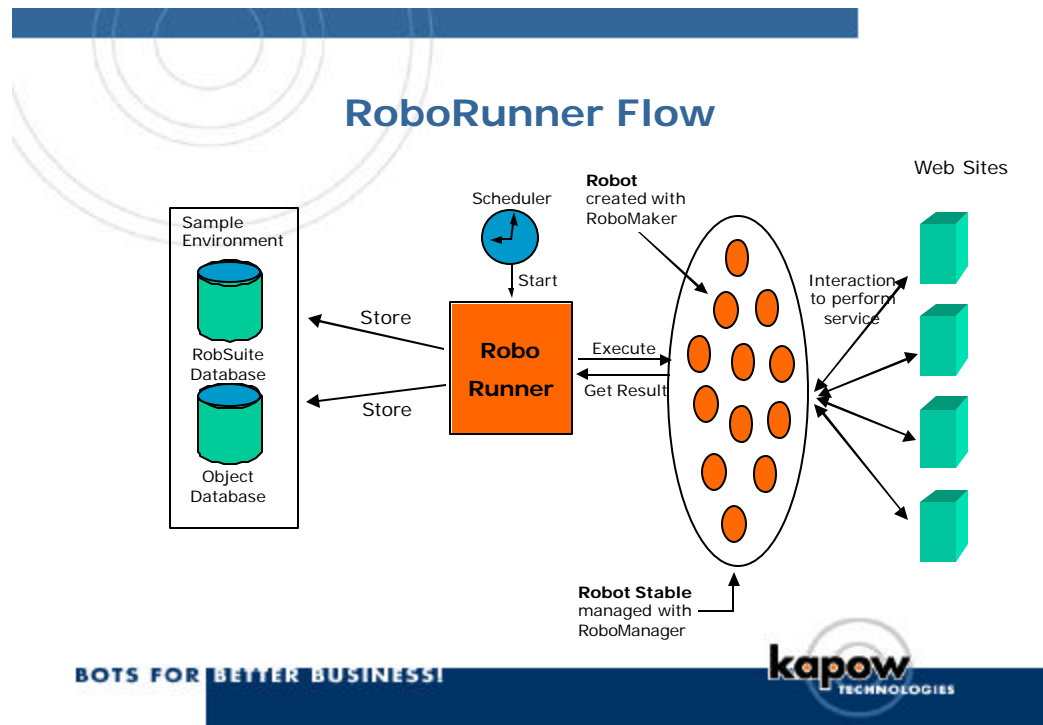
### Overview

RoboRunner is the RoboSuite application responsible for executing robots based on a command-line invocation. In conjunction with a scheduler, such as Unix Crontab, RoboRunner can run robots in batch at specific time intervals.

RoboRunner locates robots using URLs, meaning that they can be loaded across the Internet. RoboRunner can execute robots in parallel, and thereby increase the number of robots that a single machine can run. Robots are executed in highly configurable environments that determine the processing and storage of extracted objects and generated messages, robot timeout strategies, upper and lower limits on the number of extracted objects, etc. Storage environments exist for storing extracted objects in a relational database, or in a file in XML or CSV format. Message environments exist for storing robot messages in a relational database, or in a file. An email message environment can send an email when a robot fails.

## Architecture

The diagram below shows a flow-chart architecture of RoboRunner.



The concept of environments is important. Environments allow extracted objects and robot-generated messages (in addition to other things) to be pre-processed and stored exactly as you see fit, e.g. using your legacy database. Environments can be plugged in.

## Requirements

From a technical perspective, you need a computer with JRE v1.3 or later. The exact machine requirements depend on the number of robots that needs to run pr. time unit. Experience shows that a high-end PC can execute about 1,000 average-task robots pr. day. RoboRunner takes up less than 20 MB of hard disk space.

From a human resource perspective, you need a person capable of setting up a scheduler and invoking RoboRunner with the correct environment. Such a person is typically a systems administrator.

## RoboServer

### Overview

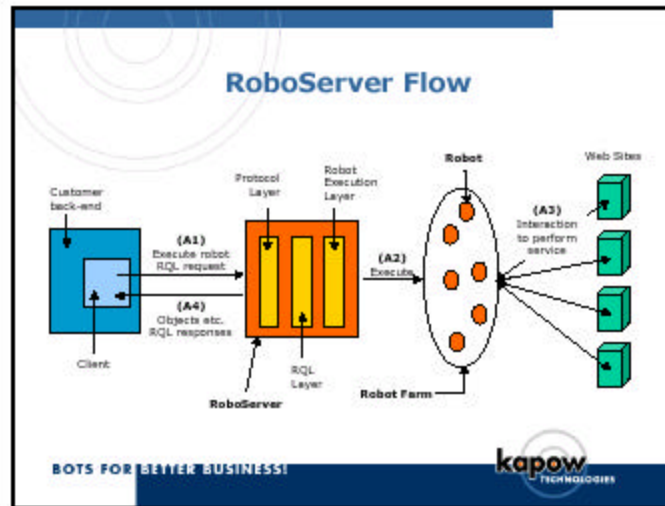
RoboServer is the RoboSuite application responsible for executing robots as a service. Clients interact with this service via RQL (Robot Query Language) represented in XML or object form. Clients are easily integrated into a customer-specific backend, allowing easy (and uniform) interfacing to robots and web services. Clients send RQL requests to a RoboServer and receive RQL responses in return.

RoboServer runs robots in an environment specified in the "Execute Robot RQL Request". This environment, setup by the client, determines e.g. how extracted objects and robot-generated messages are processed and stored. When using RoboServer, objects and messages are typically streamed back to the client that has initiated the robot execution.

RoboServer includes a sub-application, RoboClient, for testing the RoboServer and viewing the RQL requests and responses as XML (according to the RQL DTD).

## Architecture

The diagram below shows a flow-chart architecture of RoboServer.



RoboServer is built according to a layered architecture where layers can be seamlessly replaced or expanded. For example, new communication protocols, like SOAP and RMI, can be added on-demand. Communication protocols and environments can be plugged in.

## Requirements

From a technical perspective, you need a computer with JRE v1.3 or later. The exact machine requirements depend on the number of robots that need to run pr. time unit. When used as a transaction server, where the average robot-task is small, RoboServer can perform about 10-60 transactions pr. minute, depending on website response times, on a high-end PC. RoboServer takes up less than 20 MB of hard disk space.

From a human resource perspective, you need a programmer for programming the client that interacts with the RoboServer along with a systems administrator for maintaining it.

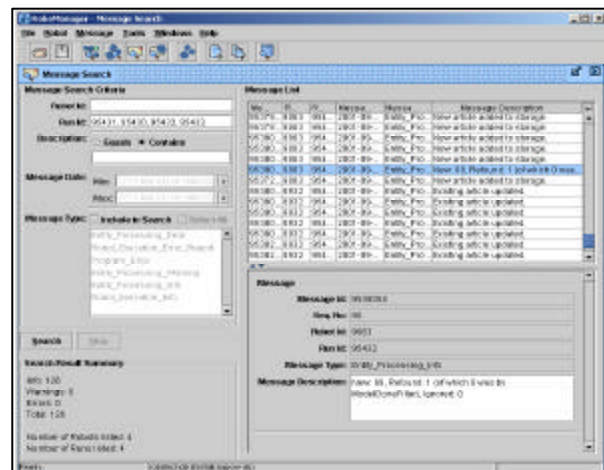
# RoboManager

## Overview

RoboManager is the RoboSuite application responsible for managing the *robot life cycle*, including robot registration, robot grouping, robot searching, robot run searching, and robot message searching. The latter three support the primary purpose of RoboManager, namely robot maintenance.

RoboManager is a database client working against a RoboSuite database with tables such as Robot, RobotGroup, RobotRun, and Message.

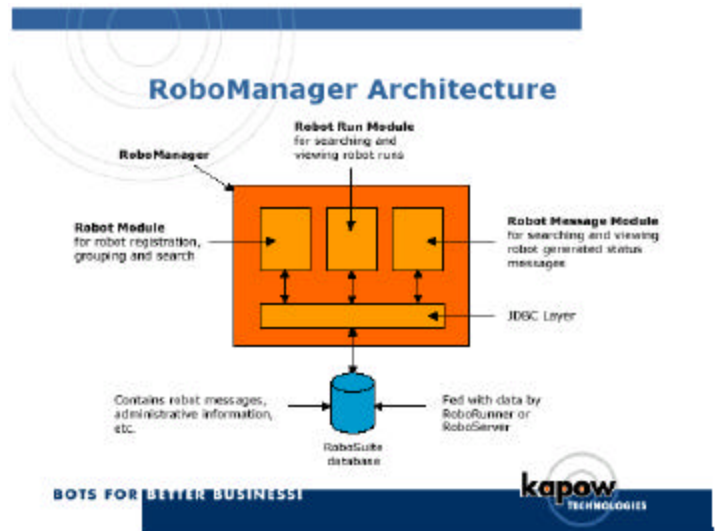
RoboManager also includes a simple GUI for running robots, and a database tool for viewing the number of database records, such as robots and messages.



## Architecture

RoboManager is a database client as shown in the diagram to the right. Not all RoboManager modules are shown in the diagram. Other modules include a simple RoboRunner GUI, and a combined robot-run-and-message search view.

RoboManager utilizes the JavaBeans component model. Each module is a JavaBean visualized in its own window allowing you to work with several modules at once.



## Requirements

From a technical perspective, you need a PC-like computer with JRE v1.3 or later. The RoboSuite database must be a MySQL, Sybase, or Microsoft SQL Server database. RoboManager takes up less than 20 MB of harddisk space.

From a human resource perspective, you need a person for managing robots using RoboManager. RoboManager is very easy to use, and typically the person that created a robot (using RoboMaker) is also the one responsible for maintaining it (using RoboManager).