The INFACT Gnome Environment: A Platform for Autonomous Agents Generating Automatic Student Assessments and Feedback

by

Nicholas J. Benson

A senior thesis submitted in partial fulfillment of
the requirements for the degree of

Bachelor of Science
With Departmental Honors

University of Washington
Department of Computer Science and Engineering

June 2004

Presentation of Work Given On: 21 April 2004

Approved By: _____
Steve Tanimoto, Project Advisor

Contents

Chapter 1

Introduction

1.1 Motivation

The majority of on-line learning environments, while fostering the collection of a wide range of student data, currently provide only a limited set of tools to aid in the analysis and assessment of data gathered by such systems. The ability to rapidly create and deploy modular applications aimed at these particular tasks is potentially very useful. With these ideas in mind, we have developed a platform for the easy creation and deployment of autonomous student data assessment agents or "Gnomes," written in Java and operating independently within INFACT, an on-line learning environment developed at the University of Washington.

The Gnome Environment consists of a development platform that eliminates the low-level concerns typically associated with programming an agent. It provides a novel system that allows for easy access to data such as web-forum posts, student sketches, logged application events, manual instructor markup and student preconception databases all housed within the INFACT system. Additionally, Gnomes are capable of receiving real-time notification events generated by the various components of INFACT.

Gnomes can be put to use in a number of different ways. They are capable of providing dynamic feedback to student activity such as web-forum posts and online multiple-choice tests. They can be used to rapidly prototype and evaluate the experimental application of advanced AI techniques to the domain of educational technology. Finally, Gnomes provide a "batch mode," allowing the system to aggregate and report on information within the environment to instructors on a periodic basis.

1.2 The INFACT Online Learning Environment

The INFACT (**I**nteractive **N**etworked **F**acet-based **A**ssessment **C**apture **T**ool) Online Learning Environment[2] provides a comprehensive set of tools to aid in the collection of student data. Each individual component of the system records extensive information that can later serve as a tracer of conceptual understanding as students progress through course material.

Built around Jim Minstrell's facet-based teaching methodology[1], INFACT boasts tools that facilitate interactive, collaborative, activity-based learning, Using the facet-based approach, the state of a student's understanding at any given point in time can be represented in terms of a set of *facets*, or common conceptions. By examining the trends present in facet distributions recorded by the system, instructors can custom-tailor curriculum to best fit varying student needs.

Key components of the INFACT system include:

- INFACT-FORUM, a threaded, web-based discussion forum, allowing students to discuss and debate solutions to course problems. INFACT-FORUM also features a "private" mode, in which posts are only visible to course instructors, providing a means by which students can submit individual work.

- INFACT-SKETCH, a Java applet integrated with INFACT-FORUM that permits students to attach free-hand sketches to forum posts. In contrast to traditional, purely text-based web-forum systems, this allows students to present their ideas visually. At the low-level, INFACT-SKETCH records and stores enough information to completely reconstruct sketching sessions, including changes and deletions, stroke by stroke.

- The Pixelmath Calculator[3], an exploratory learning tool, helps to make many of the concepts associated with the field of image processing accessible to students with a limited background in computing and mathematics.

  Additionally, the Pixelmath Calculator includes a simplified SCHEME programming environment, allowing image processing to be used as a vehicle for an introduction to the basics of computer programming.

  In a manner similar to that employed by INFACT-SKETCH, the Pixelmath calculator logs extensive event information, later allowing for detailed analysis of the actions taken by students using Pixelmath.

- INFACT-DIRECT-QUESTIONER, an online multiple choice test creation and administration system, allowing instructors to incorporate multiple choice tests into their course curriculum.

- A facetbase-management system, allowing educators to develop and maintain large facet catalogs.

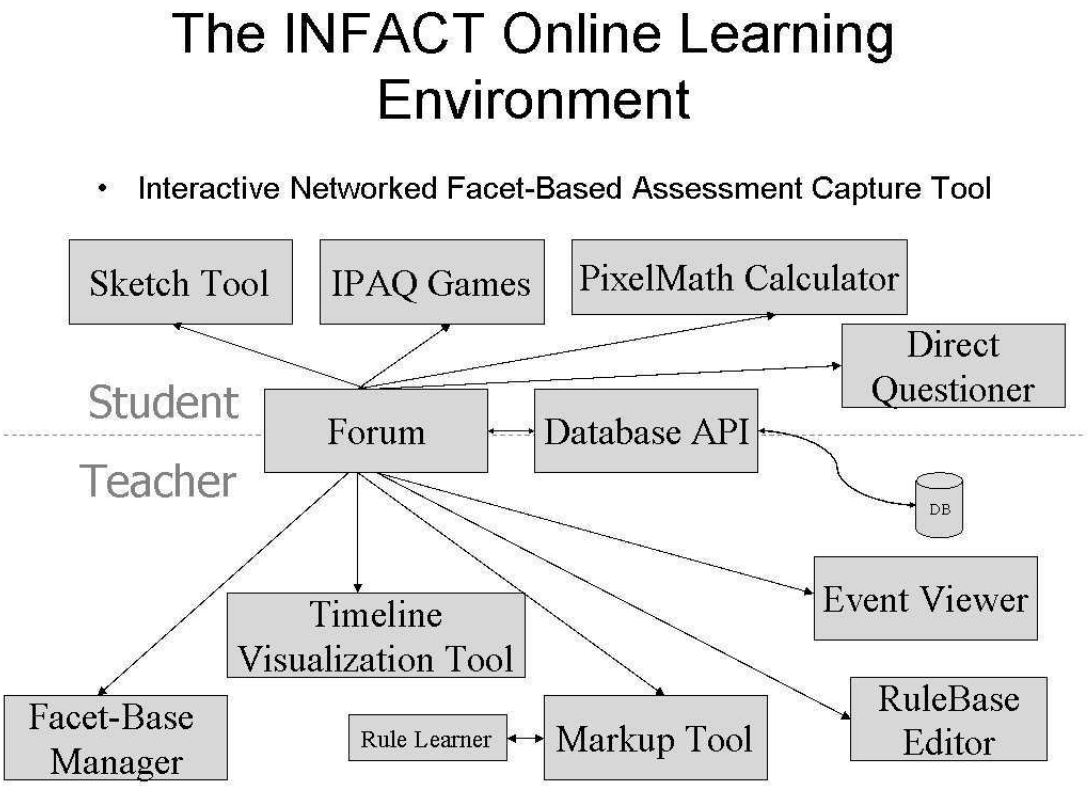
- The INFACT-MARKUP tool, a Java applet used for manually annotating INFACT-FORUM posts and INFACT-SKETCH drawings. Using the markup tool, users can highlight interesting segments of postings or sketches and associate comments, as well as facet references with such highlights.

- Various tools providing users with graphical visualizations of data generated by tools within the INFACT system.

- A generic application event-logging system allowing future INFACT developers to easily extend new system components to log information in a manner similar to that employed by INFACT-SKETCH and the Pixelmath Calculator.

The large variety and huge volume of the data collected by INFACT often makes makes manual analysis of student work in any detail an intractable task. With the goal of automating at least a part of the assessment process, the Gnomes system provides a powerful, yet easily usable interface to all of the information contained within the INFACT internal database. This, in conjunction with a real-time event notification system and a robust development environment allows developers to craft agents that reason using the full range of available information.

As a result, Gnomes have the potential to close the loop of the online learning process by automatically analyzing data collected by INFACT, classifying and annotating student activity as well as responses, and conceivably going so far as to provide students with automated feedback.



INFACT Architectural Overview

1.3 Related Work

A number of environments and standardized architectures currently exist to aid in the creation of autonomous agents[4]. These systems, while fostering agent portability and providing rich sets of features for abstracting the resources available to agents, are inherently complex, making usage with INFACT difficult. Creating an agent environment of our own allows for tight integration with the INFACT system, ease of use, and as lean and efficient an implementation as possible.

Chapter 2

The Gnome Environment

2.1 What is a Gnome?

At the conceptual level, Gnomes can be thought of as programs residing on a server, actively waiting for input. As time passes, Gnomes may receive events, either as a result of activity within the INFACT system, or by way of a pre-scheduled timer event. Gnomes are also free to query from and write records to the INFACT database. As long as the server running INFACT and the Gnome Environment remains online and healthy, Gnomes maintain their state, meaning they are capable of accumulating data from multiple sources over a period of time. Gnomes are also capable of performing tasks such as as sending dynamically created e-mail, or posting messages to the INFACT-FORUM.

At the low level, Gnomes are implemented using Java Servlet technology and the Apache Tomcat server. New Gnomes can be created by extending the `BaseGnome` class, which transparently takes care of the low-level details associated with servlets such as execution, fine-grained concurrency control and security. Implementing a Gnome is as simple as providing the implementation for a handful of methods. Individual Gnomes are capable of maintaining state by making use of the member variables defined within the classes extending `BaseGnome`. Inside the methods called during various stages in a Gnome's lifetime, developers can make use of two primary systems: the *Gnome Data Interface*, a comprehensive API providing highly abstracted access to the INFACT database as well as facilities such as e-mail, in addition to the *Gnome Events System*, an API allowing gnomes to receive real-time events as a result of real-time activity within INFACT. Additionally, developers are provided with a set of tools to ease common tasks such as file writing on the server's disk.

Using these resources, developers are able to create agents, or Gnomes, ranging in complexity from simple statistical reporting agents to complex student assessment systems making use of machine learning and other advanced AI techniques.

2.2 Modes of Gnome Execution

INFACT Gnomes operate under four distinct modes of execution, each suited to a specific type of task. It is important to note that a Gnome is free to make use of more than one mode of execution.

2.2.1 Event-Based Execution

When operating under the event-based mode of execution, Gnomes subscribe to one or more Gnome event types when the server first comes up, then sit idle waiting for events of those types to occur.

In this particular mode, all execution takes place as a result of event activity; however, typical Gnomes make use of class member variables to build up state as events are received over time.

Simple Gnomes making use of the event-based mode of execution may include an agent that dispatches e-mail to the system administrator when a certain type of event takes place within INFACT, or an agent that makes a posting to INFACT-FORUM once all members of a class have responded to a previously posted question.

2.2.2 Batch-Mode Execution

Gnomes making use of batch-mode execution are automatically run at pre-scheduled times entered by users of the Gnomes system through a web interface. Gnomes making use of batch-mode execution will typically run in one of two ways: they may accumulate state by means of the event-based mode of execution, then report information about the received events on a periodic basis, or they may periodically query the INFACT database using the Gnome Data Interface.

Simple Gnomes making use of batch-mode execution include a Gnome that e-mails forum usage statistics to course instructors on a weekly basis, or a gnome that builds and e-mails "digests," messages containing a collection of all the forum-postings made over a given period of time.

2.2.3 On-Demand Execution

The on-demand mode of execution allows Gnomes to executed immediately, invoked through a web interface. On-Demand execution can be used both to aid in the process of debugging Gnomes, as well as in cases where it makes sense to perform a certain task immediately upon request.

2.2.4 Command-Line Debug Mode

Command-line Debug Mode allows Gnomes to be invoked and run interactively from a unix shell. While the usefulness of this feature is somewhat limited when it comes to the use of agents, the ability to run and debug Gnomes from the command line greatly eases the task of Gnome development. Command-Line Debug Mode is also used extensively by developers for the purpose of extending and debugging the Gnome Environment itself.

2.3 Key Gnome Components

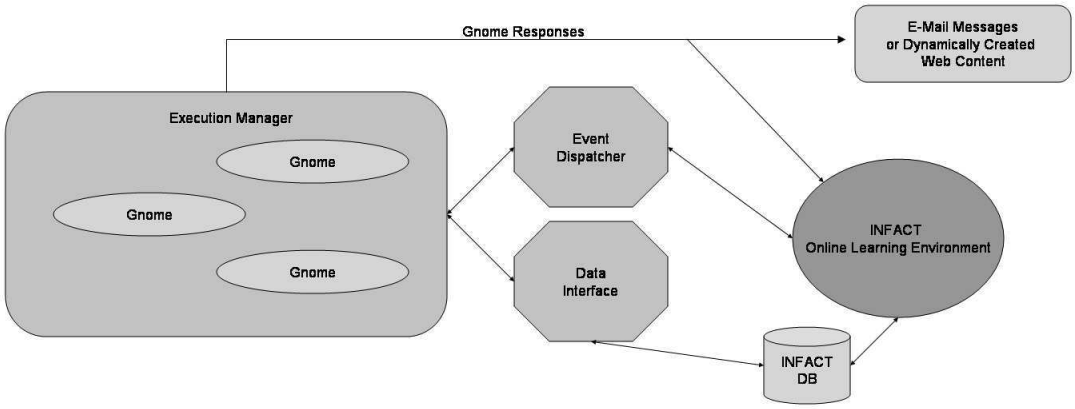
The Gnome Environment consists of four distinct components, each performing a specific task:

- *The Gnome Execution Manager*, the system responsible for scheduling and initiating the execution of Gnomes, as well as for serving as the link between the code written by Gnome developers and the Tomcat server. The Gnome Execution manager also includes a web interface that allows users to schedule, manually invoke and monitor the health of the Gnomes already installed on the server.

- *The Gnome Event Dispatch and Management System*, an extensive API that manages Gnome event subscriptions and dispatching.

- *The Gnome Data Interface*, which serves as the primary link between Gnomes and the INFACT database, as well as the outside world. It provides clients with the ability to query and update the INFACT database, in addition to allowing users to perform tasks such as sending email and posting to the INFACT-FORUM.

- *The Rapid Gnome Development Toolkit*, a series of command-line utilities to ease the task of creating, installing and maintaining new Gnomes.



Basic Architecture of the Gnome Environment

2.4 The Gnome Execution Manager

The Gnome Execution Manager comprises the core of the Gnome Environment. The Execution Manager includes the code that abstracts the low-level details of servlet-Gnome interaction, allowing developers to concentrate solely on the task of creating an agent.

2.4.1 Gnomes and Servlets

Why Java Servlets?

Early on in the design of the Gnome Environment, the decision was made to leverage Java Servlet technology, as opposed to implementing the low-level functionality of an agent execution management system from scratch. Given the limited amount of time available for the implementation of the system, this allowed us to devote more effort to the task of creating a system well-suited toward the task of applying agents to the realm of educational technology.

Gnome-Servlet Correspondence

INFACT Gnomes are intimately related to Java Servlets. At the Java implementation level, a Gnome is nothing more than an extended HTTP servlet class making use of a set of sophisticated APIs. Developers build new Gnomes by extending the abstract class `BaseGnome`, implementing methods that are called when the Gnome Execution Manager starts up or shuts down, and when various events corresponding to the various modes of execution detailed above occur (Please consult the Gnome Development Guide for a more detailed explanation.)

Gnomes (and Servlets in general) are automatically loaded into memory when the Tomcat server first starts up. They then begin execution of an initialization method when woken up by an HTTP request (when installed on the Tomcat server, each servlet has a unique URL through which HTTP connections can be established.) After initialization has taken place, Gnome activity can be established in one of the following manners:

1. *HTTP Requests.*
   By default, when an HTTP GET or PUT request is made to a Gnome, the system returns

a web page displaying simple information such as the name of the Gnome with which a connection has been established, the current time and the number of requests made to the Gnome since the Tomcat server last came up.

The `HTTPServlet` class within Java Servlet API provides methods for handling all HTTP request types, each of which can easily be overridden by Gnomes requiring such special functionality. This is, however, an unsupported feature. Please refer to the J2EE API reference for more information on servlets and HTTP functionality.

2. *The Gnome Environment Servlet Context Hash.*
   As the name implies, the Gnome Environment Servlet Context Hash provides a `Hashtable` that is shared between all Gnomes operating within the Gnomes environment on a given server. A common means of communication between collaborative Gnomes is to have multiple Gnomes acting as data producers and consumers, reading and writing from the Servlet Context Hash in parallel. It is important to note that the Servlet Context Hash is automatically synchronized, eliminating the need for developers to worry about race conditions.

3. *Gnome Events.*
   Gnome Events provide a means by which Gnomes can remain apprised of the happenings within the INFACT environment. The Gnome Environment also uses Gnome Events internally to start the execution of Gnomes in batch execution mode.

Java servlet technology also provides Gnomes with the ability to write to file systems mounted on the host running the Tomcat server. Gnome file I/O by Tomcat processes at runtime, and as a result, is subject to the read/write permissions of the account under with Tomcat is run.

Gnome Persistence and the Multiple-Thread Execution Model

When the Tomcat server first comes up, all Gnome classes installed on the server are loaded into memory. The constructor for each Gnome is called at this time, however, as a matter of style, it is not common for Gnomes programmers to implement a constructor for Gnome classes, but rather to place the code that would normally reside there within the `startupGnome()` method. Again, please consult the Gnomes Development tutorial for more detailed information.

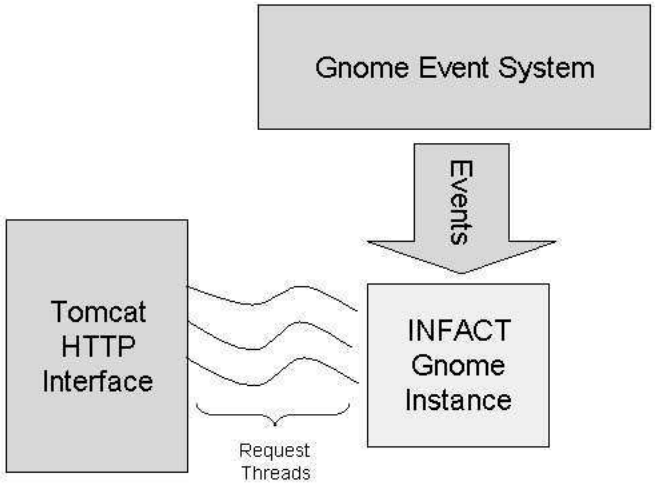
The first time an HTTP request is made to a Gnome, `initializeGnome()` is called, followed by the handler method for the relevant HTTP request type if it has been overridden by the Gnome class.

From this time until the Tomcat server goes down, the Gnome class remains resident in server memory. All member variables and the corresponding objects they reference remain intact. This allows Gnomes to build up and maintain state as Gnome Events are received and HTTP requests are made.

When the Tomcat server is shut down, the `shutdownGnome()` method of each Gnome class is called, allowing the Gnome to perform any necessary operations such as writing data to the database or disk and to free resources. It is important to note that if Tomcat is not shut down in an orderly manner (for example as a result of the Tomcat processes being killed or a power outage), `shutdownGnome()` will not be called, and data may subsequently be lost.

Throughout the duration of the time the Tomcat server remains online, only a single instance of each Gnome class is instantiated. Concurrent requests to a single Gnome result in multiple threads accessing the single instance of that Gnome's class simultaneously. This is referred to as the *Multiple Thread Model*. As a result, great care must be taken in Gnome development to ensure

that all Gnome Event and HTTP Requests are reentrant (e.g. all instance variables and resources referenced within these methods must be synchronized.)



The Multiple Thread Model

The Gnome Persistent Data Store

One key problem with Java Servlets resides in the fact that all data stored within the member variables of a Servlet class is lost when the Tomcat server is shut down. This is problematic for Gnomes, as it is often desirable for Gnomes to maintain their state indefinitely. As a solution to this problem, the Gnome Environment includes a feature called the "Persistent Data Store."

The Persistent Data Store consists of a special member variable, `persistentDataStore`, which points to a `Hashtable` that persists even when the Tomcat server goes down. This "super-persistence," is accomplished by code that automatically serializes the persistentDataStore hashtable and writes it to disk when Tomcat goes down, and reads back and de-serializes the hashtable information when the server comes back up.

In this manner, it is possible for Gnomes to maintain data indefinitely, without programmers having to manually write data to disk or the INFACT database.

The Gnome Logging System

In the course of developing and debugging Gnomes, it is often useful to have the ability to output information to some sort of error log. The Tomcat server provides this functionality by allowing users to write to a standard error log when calls to `System.err.print()` are made. However, the information within this log is completely unstructured, and in cases where multiple Gnomes are running concurrently, the log becomes nearly indecipherable.

In response to this problem, Gnome classes automatically inherit a `log(String str)` method, which writes error messages to an error log containing only the error messages for the Gnome from which it's called. Within the Gnome Environment directory on the server, a subdirectory exists that contains a log file for each Gnome installed on the system.

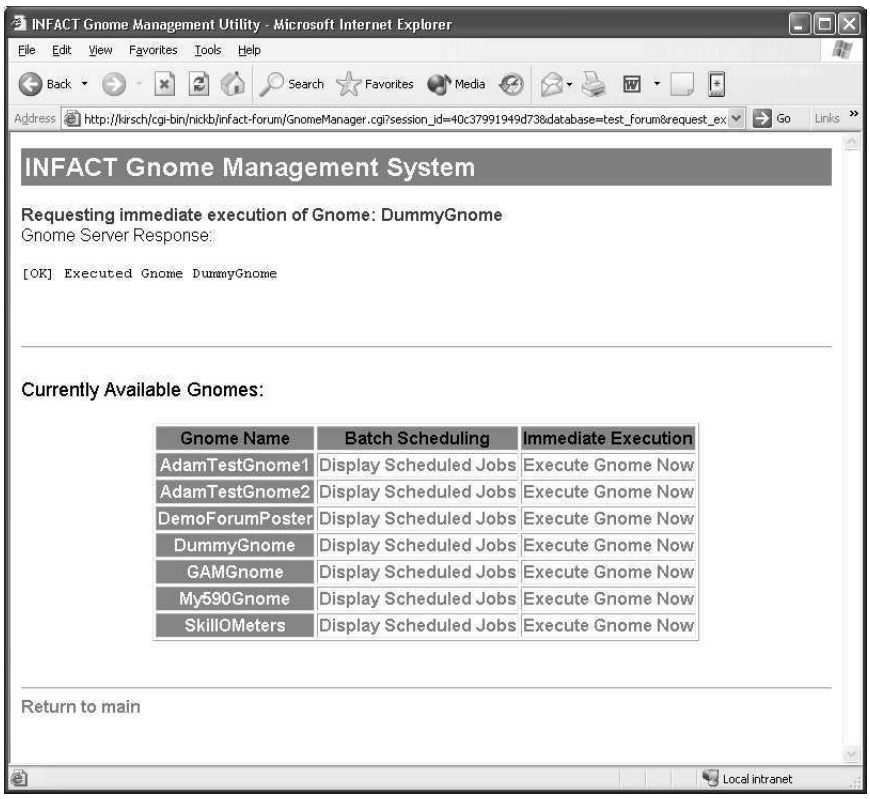
Should a Gnome developer wish to automatically prefix each line of a Gnome's error log with a timestamp, the inherited `enableLogTimestamps()` and `disableLogTimestamps()` methods toggle this functionality.

2.4.2 The Gnome Environment Web-Interface

The Gnome Environment web-interface provides users with an easy to use means of requesting immediate execution of Gnomes and managing the execution schedules for Gnomes running in batch-mode. The Gnome web-interface tool is available under "Visualization and Assessment Tools" from within INFACT-FORUM.

Usage

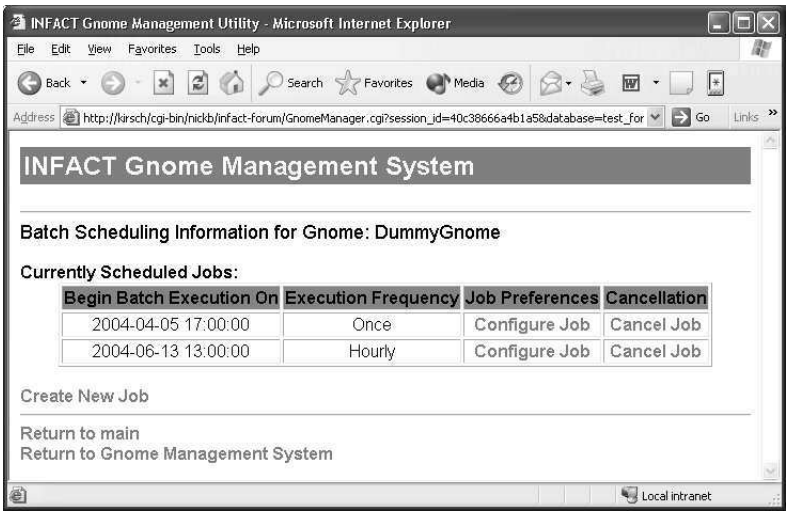
When the Gnome Environment web-interface is loaded, it displays a list of the Gnomes currently installed on the system. From this screen, users have the option of either requesting immediate execution or displaying the batch-mode scheduling information for a particular Gnome.



Executing a Gnome Through the Web Interface

When the user clicks on an *Execute Gnome Now* button, the web-interface contacts the Gnome Environment and calls the appropriate Gnome's `executeImmediate()` method. The interface then displays the response sent by the Gnome Environment indicating whether or not execution was initiated successfully.

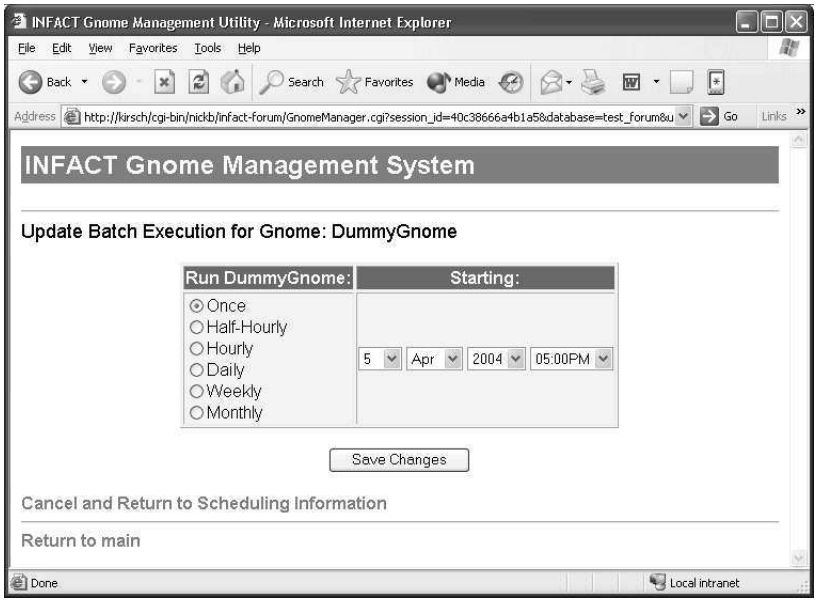
Users can view batch-mode scheduling information by clicking on the *Display Scheduled Jobs* link for any Gnome of interest. This opens a new page displaying the times for which batch-mode execution of the Gnome has been scheduled.



Scheduling Information for an Individual Gnome

From this screen, users can schedule a new execution time, remove a time that has previously been scheduled, and update the preferences for an existing time.

When creating or modifying a batch-mode execution time, users must specify two parameters: the date and time when batch-mode execution of the Gnome should first take place, and how frequently (if ever) execution should be repeated.



Scheduling Preferences Configuration

Implementation

Internally, the Gnome Environment web-interface is implemented using Perl CGI scripts. When immediate execution of a Gnome is requested through the interface, the system dispatches an execution request event to Gnome Event Dispatcher, which responds by invoking the appropriate Gnomes' `executeImmediate()` method.

The web-interface makes use of a SQL database for storing batch-mode execution scheduling information. Whenever the scheduling database is updated, the system dispatches an event to the Gnome Event Dispatcher, which in turn consults the database to update the scheduling management data structures for the relevant Gnomes.

To guard against unauthorized access to the Gnome Environment, the interface requires users to authenticate using an INFACT username and password before using the system.

2.5 The Gnome Event Dispatch and Management System

The Gnome Event Dispatch and Management system takes care of all issues relating to the management of event subscriptions as well as the definition, production and dispatch of Gnome events. The system consists of two key components: a module within the Gnome Environment that produces and dispatches events in response to activity within INFACT, and a series of modifications to the INFACT Online Learning Environment itself.

2.5.1 Design Concerns

In designing the Gnome Event Dispatch and Management System, one of the key problems to solve was the determination of a way to efficiently notify Gnomes of happenings within the INFACT environment in real or near-real time. The two main proposed solutions consisted of database-polling and active-notification event dispatch strategies.

Using a database-polling approach, the various modules of INFACT would write event data directly to a relational database. At the same time, the Gnome event system, polling the database for new event information, would proceed to notify all subscribed event listeners as soon as new event activity is detected. The main benefit to a polling-based event notification approach lies in the fact that it requires no modification to the existing INFACT codebase. The costs associated with a polling-based system include a relatively large amount of server CPU time that is wasted in the polling process, in addition to a substantial latency between the time an event actually occurs in INFACT and the time that the Gnome system becomes aware that the event has taken place.

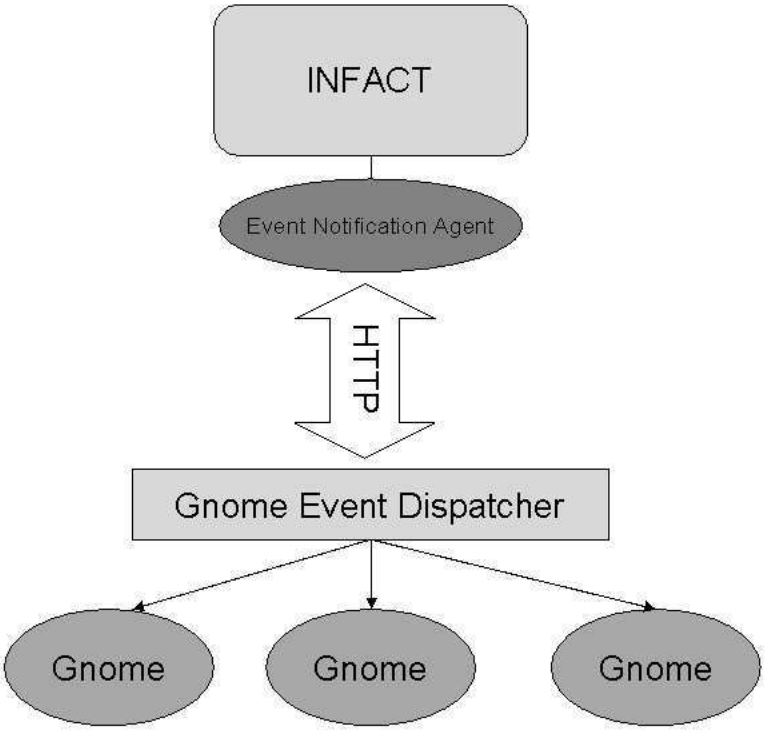
In the end, we opted to go with the active-notification approach, realizing the high-frequency nature of many commonly used event types demanded a level of performance that would be unattainable using a polling scheme.

2.5.2 Event System Implementation

When an event takes place within INFACT, the following sequence of events typically transpires:

1. The event occurs.
2. An HTTP agent, already resident in memory and part of INFACT, contacts the *Dispatcher* servlet sending it data on the event that just took place, including such information as the event-type, the forum in which the event occurred, as well as various other event-type specific values.
3. The Dispatcher servlet receives the information and creates a *Master Event Object* from the received data
4. Once the *Master Event Object* is created, the dispatcher consults its subscribed-client list and dispatches copies of the master event to each subscriber.

5. Subscriber Gnomes receive the event and respond accordingly.



The Gnome Event Dispatch Process

Modifications to the INFACT Environment

In updating the INFACT Environment to support Gnome Event dispatching, one of the main goals was minimizing the number of changes to the existing code to avoid polluting the codebase.

All changes to INFACT were made within the Perl modules responsible for providing functionality such as database access and user authentication. Modifications include the addition of HTTP agent objects (using the HTTP::Agent module available through CPAN) to each module, in addition to code that uses the agent objects to contact the Gnome Event Dispatcher with event information as appropriate. HTTP PUT parameters are used to encode event information.

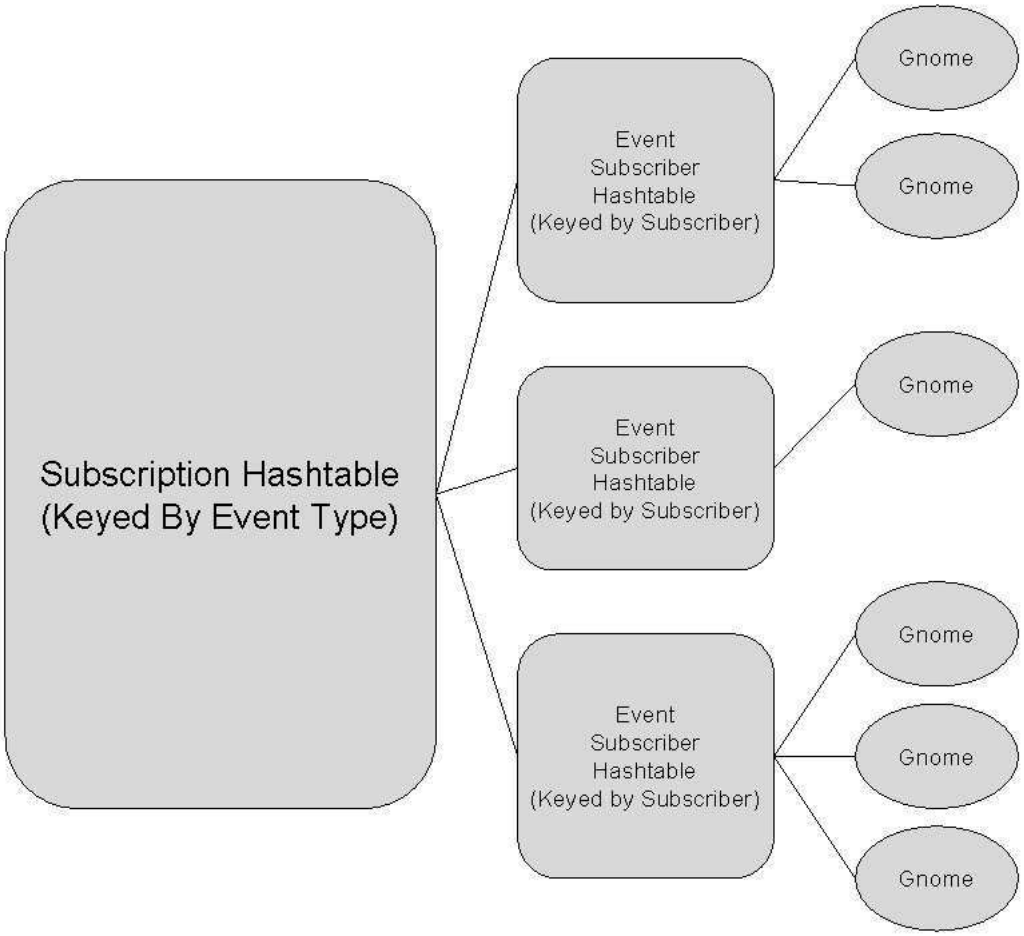
As an added feature, all Gnome-specific additions to INFACT can be disabled by setting an *ENABLE_GNOME_EVENTS* variable to "false" within the INFACT configuration Perl module. The URL of the Gnome Event Dispatcher can be set and changed in a similar manner.

The Dispatcher Servlet

The Dispatcher Servlet, automatically loaded when the Gnomes Environment is started up, sits idle waiting for event notifications through HTTP requests, as well as for event-subscription requests from client Gnomes.

As individual Gnomes are loaded by the system, they make calls to the Dispatcher, subscribing themselves to the various available event types. This process amounts to the Dispatcher storing a reference to the subscribing Gnome within a hashtable of subscribers for the appropriate event

type. To simplify event subscription for Gnome developers, a `subscribeToGnomeEvent` method is provided, taking only the event subscription type as a parameter.



The Gnome Event Subscription Tracking Data Structure

When notification of a new event is received through the Dispatcher servlet's HTTP interface, the Dispatcher first validates the request parameters. These parameter values are then passed to the `GnomeEventFactory`, a system that parses the event parameters and creates a *Master Event Object* containing the received event information.

Once the *Master Event Object* has been created, the Dispatcher consults its subscription tracking data structure in order to determine which, if any, of the currently active Gnomes need to be notified of the new event. A list of subscribers is compiled, and clones of the *Master Event Object* are distributed to each listening Gnome.

If an error occurs at any time in the event creation and dispatch process, the Dispatcher aborts its current task and writes a message to the Tomcat error log.

2.5.3 Supported Gnome Event Types

The following list provides a brief overview of the various types of events currently supported by the Gnome Environment. Please keep in mind that this list is expanding and changing as development continues. Consult the Gnome Environment Javadoc for more detailed and up-to-date information.

Forum Post Events

Forum Post Events are generated as a result of posts being made to the INFACT-FORUM. They contain information on the forum and group to which they were made, as well as the identifier of the message posted, for retrieval from the database.

New Infact Generic App-Session Events

New Infact Generic App-Session events are created when users first open on of the event-logging applications such as the Pixelmath Calculator or the Sketch tool within INFACT. New Infact Generic App-Session events contain all the information necessary to retrieve detailed information about the App-Session from the INFACT database.

Generic INFACT Application Events

Generic INFACT application events correspond to the fine-grain events generated by event-logging INFACT applications. They contain sufficient information to determine the user and application that generated them, as well the identifiers necessary to retrieve detailed information about the events from the database.

Generic INFACT Application Events are distinguished by the fact that they are the most frequently occurring event-type in the system (under a heavy load, many thousands of these events may be received daily,) so attention should be paid to efficiency when implementing Generic INFACT Application Event listeners.

DQ Response Submitted Events

DQ Response Submitted Events are generated when users provide an answer to a multiple choice test question through the INFACT Direct Questioner Tool. DQ Response Submitted Events contain information identifying the user who made the response, the response itself, and the Direct Questioner test and identifier for the question to which the response was made.

Text-Markup Creation Events

Text-Markup Creation Events are generated when users save text highlights within the INFACT-MARKUP tool. They contain the forum to which the highlight belongs in addition to the identifier necessary to retrieve more detailed information about the highlight from the database.

Text-Markup Removal Events

Text-Markup Removal Events are created when text highlights are removed from a database. The contain only the id and database name corresponding to the highlight that was removed.

App-Session Markup Creation Events

App-Session Markup Creation Events are generated when app-session highlights (such as INFACT-SKETCH highlights) are saved from within the INFACT-MARKUP tool. They contain the forum to which the highlight information belongs, in addition to the identifier necessary to retrieve more-detailed highlight information from the database.

App-Session Markup Removal Events

App-Session Markup Removal Events are created when app-event highlights are removed from a database. Like Text-Markup Removal Events, they contain only the id and database name corresponding to the highlight that was removed.

Assessment Creation Events

Assessment Creation Events are generated when assessment records are saved from within the INFACT-MARKUP tool. Assessment Creation Events contain the database to which the assessment was saved, in addition to the assessment ID. This allows users to retrieve the assessment record from the database, as well as to obtain any highlights or facet associations liked to the assessment record.

Assessment Removal Events

Assessment Removal Events are generated when an assessment record is removed from a database. They contain only the ID and database name corresponding to the assessment record that was removed.

Facet Association Creation Events

Facet Association Creation Events are generated when a user associates a particular facet with an assessment record using the INFACT-MARKUP tool. This event occurs when the assessment is saved to the database, not immediately after the user makes the association within the markup tool. Facet Association Events contain the identifier of the facet association and database to which the association was saved, in addition to the facet database and facet identifier of the facet involved in the association.

Facet Association Removal Events

Facet Association Removal Events are generated when a facet association is removed from the database. They contain the ID of the association and the name of the database from which it was removed.

User Successfully Validated Events

User Successfully Validated Events are generated when a user successfully authenticates with the INFACT system. They contain the username of the user who authenticated, the forum into which they logged in, and the time at which the authentication occurred.

2.5.4 Gnome Events and Batch/Immediate Gnome Execution

Batch- and Immediate-Mode Gnome Execution is made possible by a series of special internal events, created the the AutoScheduler servlet and received by the Dispatcher. Rather than passing these special events on to subscribed Gnomes, the Dispatcher automatically executes the action associated with the internal event type, such as immediately executing a Gnome.

Currently supported internal Gnome events include:

Immediate Execute Requested Events

The Immediate Execute Requested event is dispatched to a Gnome when immediate execution of that Gnome is requested through the Gnome Environment Web Interface. As a result of this event, the Gnome Event Dispatcher automatically calls the `immediateExecute()` method of the Gnome to which the event referred.

Batch Execute Requested Events

The Batch Execute Requested event is dispatched to a Gnome when batch-mode execution of that Gnome is requested by the Auto Scheduler. This typically happens as a result of an internal timer event. When a Batch Execute Requested event is received, the Gnome Event Dispatcher automatically calls the `batchExecute()` method of the Gnome to which the event referred.

2.6 The Gnome Data Interface

The Gnome Data Interface is an extensive API sitting atop Java's JDBC system allowing users to easily obtain data stored within the INFACT relational database, in addition to performing tasks such as sending dynamically created e-mail messages.

The Gnome Data Interface was created especially for use within Gnomes. However, the system has proved robust enough for use within other Java applications residing on the server running the Gnomes environment.

At the time of writing, the Gnome Data Interface is still in a state of development. Therefore changes are being made and new features added on a very frequent basis. Please consult the Gnome Data Interface Javadoc for detailed and up-to-date information on the system.

2.6.1 General Discussion

At the top level, the Gnome Data Interface consists of three primary components: a system for establishing and caching connections to INFACT database(s), a series of content "getters" for easily retrieving information from the databases, and a series of content "putters" for the creation of new content.

As the Gnomes Data Interface relies on a number of distributed resources such as database and network connections, the throws a number of different exceptions. Again, please consult the Javadoc for a more explicit treatment of the types of exceptions that are thrown, and those that must be caught when various components of the Gnome Data Interface are put to use.

2.6.2 Using the Gnome Data Interface

DataInterface Objects

The first step to using the Gnome Data Interface system is obtaining a `DataInterface` object. `DataInterface` objects are used to access all components of the system.

Establishing Database Connections and Obtaining GDI Content Objects

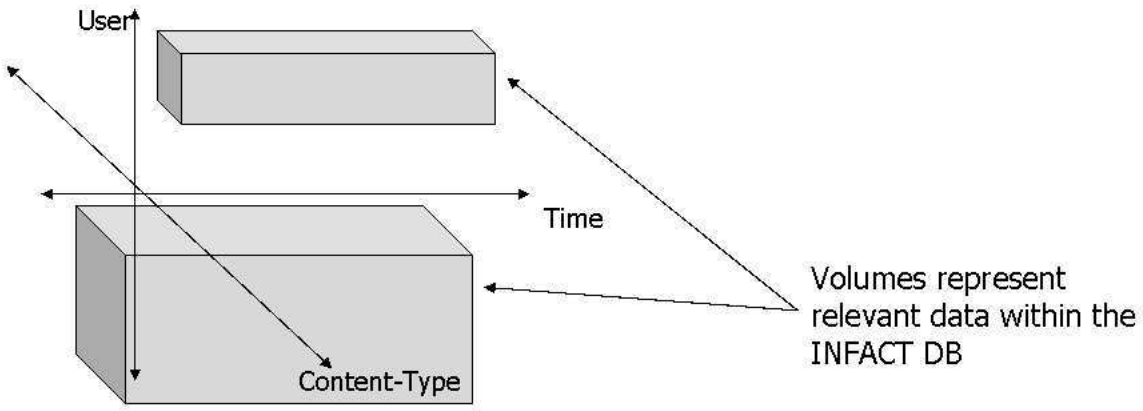
In an effort to improve efficiency, the Gnome Data Interface makes use of a sophisticated database connection caching system. As a result, before reading or writing content to INFACT, users must first establish connections to the databases they plan to access. Internally, database connections

are stored within a hashtable keyed by database name, and persist until such time as they are either explicitly destroyed or the server goes down. Gnome Data Interface database connections handle synchronization (e.g. instances where multiple threads attempt to access the same database connection handle) automatically.

Once database connections have been established, content can then be read from or written to the system by way of *data fetchers* and *data putters*, objects used for retrieving and storing content, respectively. `DataInterface` objects contain a series of methods for creating new getters and putters, one corresponding to each available content type. Content getter and putter creation methods take a database connection identifier as a parameter.

Querying using the GDI: Getters and "Data Volumes"

Aiming to provide users with fast, uniform access to INFACT data, *Getter Objects* allow users to build up "Query Objects," selecting data using "volumes" in a high-dimensional space as an analogy for the set of matching data. Under this paradigm, volume boundaries are defined by imposing additional query constraints.



A graphical representation of Getter query constraints

As an example, a user may choose to select all the INFACT-FORUM posts within a particular forum over a certain time range, including only those posts made by administrators, and excluding posts made during weekends. In this case, selecting only posts defines a constraint on content, the time range and exclusion of posts during weekends defines two constraints on time, and the the restriction of returned posts to only those written by administrators defines a constraint on content authors.

At the API level, each content getter object contains a series of "include" and "exclude" methods, allowing client code to build up the state of a query by imposing multiple constraints. Once the getter reaches its desired state, a `retrieveResults()` method can then be called, automatically generating a SQL query from the constraint information, hitting the database, parsing the results and returning a vector of matching content objects.

Creating Content: Putter Objects

For the purpose of creating content, the Gnome Data Interface provides a series of *Putter* objects. Simpler in structure than Getter objects, Putter objects function by first having users instantiate and assign values to ADTs corresponding to content-types within INFACT,

2.6.3 Generic API Datatypes

The Gnome Data Interface defines the following generic datatypes to aid in the creation of database queries:

GAutoMessage

`GAutoMessage` objects are used to represent e-mail messages within the GDI. They include fields for specifying message authors, recipients, CC and BCC recipients, as well as fields for the message subject and body.

GDateRange

`GDateRange` objects are used for the purpose of specifying time periods when creating Getter object query constraints. They contain four primary fields: a start date and time, and an end date and time.

GTimestampRange

`GTimestampRange` objects are used for the same purpose as GDateRange objects, however, they use the SQL `Timestamp` datatype, as opposed to separate date and time fields.

GIntRange

`GIntRange` objects are used to represent ranges of integers when forming query constraints. Example applications of this datatype include the specification of a range of INFACT-FORUM posts, or app-event IDs.

2.6.4 INFACT-Specific API Datatypes

Content read from and written to the INFACT system is typically represented within the Gnome Data Interface using the following set of INFACT-specific datatypes:

InfactAppSession

`InfactAppSession` objects are used to represent INFACT application sessions. They contain information on the session author, creation time and session content-type.

InfactAppEvent

`InfactAppEvent` objects correspond to the individual application events stored within the INFACT database. They include information on the date and time at which the event occurred, the user who generated the event, the application responsible for generating the event, as well as a free-form field in which event-specific information is stored.

InfactForumMessage

`InfactForumMessage` objects correspond to postings made within INFACT-FORUM. They contain information such as the author, subject and body of the posting, the group(s) to which the message was posted, the date and time of the post, and the internal ID used to track the post within the database.

InfactForumRulebase

`InfactForumRulebase` objects are used to store rule-bases: sets of text-matching rules that are created and managed by the INFACT Rule-base Editor tool. `InfactForumRulebase` objects consist of one or more groups of `Rule` objects, each of which includes a text-matching pattern similar to a regular expression in addition to an action field that specifies the steps to be taken when the rule's pattern is matched.

RuleSchedule

`RuleSchedule` objects contain information used to schedule the automatic execution of INFACT Rule-bases. `RulebaseSchedule` objects contain information on which rule-base to be execute, the time(s) at which the execution should take place, and the set of user group(s) to which the rule-base should be applied.

InfactForumUser

`InfactForumUser` objects encapsulate data about users of the INFACT system. They include fields for representing the user's first and last names, username, account type (user, persistent-user or administrator), the user's internal id and last login date.

2.6.5 Content Fetcher Objects

The following Content-Fetcher objects are currently supported by the Gnome Data Interface. Several more are currently under development.

PostFetcher

`PostFetcher` objects are used for the purpose of retrieving INFACT-FORUM posts from the data base. They allow users to filter posts by defining constraints on parameters such as regular expressions matching the post subject and body, post time, author, group and post internal ID.

UserFetcher

`UserFetcher` objects are used to retrieve information about user accounts within the INFACT system. Account records can be filtered by account first and last names, username, internal ID, last login date, last active date, and group ID.

AppSessionFetcher

`AppSessionFetcher` objects are used to retrieve application event session records from the database. `AppSessionFetcher` queries can be filtered by session ID, session creation type, the creator of the app session, as well as by the app-session type. If desired, `AppSessionFetcher` objects can also automatically retrieve all of the individual events associated with each matching app session.

EventFetcher

`EventFetcher` objects allow client code to retrieve information on individual application events. Event queries can be filtered by internal event ID, event type, parent app-event session type, event author, event-specific content and event creation date.

RuleBaseFetcher

`RuleBaseFetcher` objects allow users to retrieve rule-base information from the INFACT database. Rule-base queries can be filtered by rule-base ID, author, name and creation date.

RuleScheduleFetcher

`RuleScheduleFetcher` objects provide a means of retrieving rule-base execution scheduling information. Queries to retrieve rule-base scheduling information can be filtered by the name or ID of the rule-base to be executed, schedule record ID, schedule record owner and scheduled execution time.

DQResponseFetcher

`DQResponseFetcher` objects allow for easy retrieval of responses to INFACT-DIRECT-QUESTIONER responses made by users. Response queries can be filtered by respondent, response time, the question to which the response was made, the answer choice that was made, as well as by internal response ID.

2.6.6 Content Putter Objects

The following set of Content-Putter objects are currently supported by the Gnome Data Interface:

MessagePutter

`MessagePutter` objects are used for the purpose of programatically making posts to the INFACT-FORUM. Typically, users first create one or more `InfactForumMessage` objects, the use a `MessagePutter` object to post the content to the system. Users have the option of either explicitly specifying the username under which the message should be posted or using a special "Auto Notifier" account.

EventPutter

`EventPutter` objects allow client code to write events directly to the database. It is important to note that when events are generate through the Gnome Data Interface, they are *not* logged by the Gnome Event System. This feature was put into place to help prevent feedback cycles from occurring within Gnomes.

AutoMailPutter

`AutoMailPutter` objects are used to automatically send e-mail messages. In a manner similar to that employed by `MessagePutter` objects, users have the choice of sending email using either a name and return-address specified at run-time, or a default "Auto Notifier" e-mail account. Sending e-mail using an `AutoMailPutter` is as simple as creating a `GAutoMessage` object, then calling the Mail Putter's `send()` method.

RuleSchedulePutter

`RuleSchedulePutter` objects allow automatic rule-base execution to be scheduled programatically. This feature is used by first creating a `InfactForumRuleBase` object, then calling the schedule putter's `save()` method.

2.6.7 SQL Content Objects

In the process of writing Gnomes where more complex database queries are required, situations arise where the abstraction provided by the Gnome Data Interface makes creation of such queries impractical, if not impossible. For such situations, the GDI provides `SQLGetter` and `SQLPutter` objects that allow users to access the database using raw SQL queries.

SQLGetter Objects

`SQLGetter` objects operate in a manner similar to other Getter objects, differing in that they support the imposition of only a single constraint: a SQL `SELECT` statement. When an `SQLGetter`'s `returnResults()` method is called, a `Vector` of `Hashtable` objects is returned, where each `Hashtable` corresponds to a row returned by the query, and each key/value pair corresponds to a column.

SQLPutter Objects

`SQLPutter` objects allow users to issue `INSERT`, `UPDATE` and `DELETE` statements to INFACT databases. They are used by first specifying a SQL statement, then calling a `commit()` method that executes the statement.

2.6.8 Unit Tests

To aid in the development and debugging of the system, the Gnome Data Interface includes a series of unit tests exercising the various components of the system. In addition to verifying that the GDI is working correctly, they also provide example code demonstrating the techniques that are typically employed when using each of the different Getter and Putter objects. Please consult the INFACT developer wiki web for more information.

2.7 The Rapid Gnome Development Toolkit

The INFACT Gnome Environment was designed explicitly for use by developers without detailed knowledge of the inner-workings of INFACT or the Gnomes Environment itself. As a result, the system includes a series of shell scripts to aid in the creation, installation and removal of new Gnomes.

2.7.1 Toolkit Scripts

create_gnome_template

The `create_gnome_template` script generates a skeleton Java source file for new Gnomes. Taking the name of the new Gnome being created as a parameter, it generates an appropriately named and commented Java class using a templatized "master source Gnome" file. The skeleton .java file contains empty methods which must be implemented when creating a new Gnome, in addition to extensive comments to aid new Gnome developers.

register_gnome

The `register_gnome` script allows developers to reserve a names for their Gnomes, primarily for the purpose of preventing naming conflicts in multiple-developer environments. The `register_gnome`

script takes a single parameter: the gnome name to be registered. If the Gnome name is available, the script registers the new new name and returns success. If a Gnome using the requested name already exists within the system, an error code is returned. Valid Gnome names conform to the same rules as Java class names.

Internally, the `register_gnome` script is implemented using a series of lockfiles residing within a special directory on the Gnome Environment server, one file for each registered Gnome name.

unregister_gnome

The `unregister_gnome` script functions in a manner similar to `register_gnome,` taking the name of the Gnome that the user wishes to free as a parameter. After verifying that the lockfile for the Gnome to be removed is owned by the user calling the script, `unregister_gnome` then proceeds to remove the lock, making the name available for use by other users.

wakeup_gnome

The `wakeup_gnome` script. taking as a parameter the name of the Gnome to be awakened, contacts that Gnome via an HTTP request and prepares it to begin receiving events and executing on demand.

wakeup_all

The `wakeup_all` script, taking no parameters, calls `wakeup_gnome` on each currently installed Gnome. This particular script is especially useful when the Tomcat server running the Gnome Environment has been restarted, and all gnomes need to be woken up.

make_gnome

The `make_gnome` script is used to generate the source tree for a new Gnome. Taking the name of the Gnome to be created as a parameter, the script first creates a directory for the new Gnome in the user's current working directory. It then makes a call to `create_gnome_source`, generating the .java source file for the new Gnome. Finally, it creates a Makefile, in addition to a directory where auxiliary Java code accessed by the Gnome is to be placed.

Once `make_gnome` has been called and the developer has completed work, the new Gnome's source can be compiled by running `make`, and installed by running `make install`. The `install` target of the makefile checks to make sure the compiled Gnome is up to date, copies the Gnome, along with any auxiliary files to the server installation directory, restarts Tomcat and wakes up all Gnomes installed on the server.

remove_gnome

The `remove_gnome` script, taking the name of the Gnome to be removed as a parameter, takes all the necessary steps to uninstall an already deployed Gnome. After verifying that the Gnome to be removed belongs to the user who issued the `remove_gnome` command, the script halts execution of the Gnome if running, removes the files associate with the Gnome from the Gnome environment and un-registers the name associated with the Gnome being removed.

Chapter 3

Gnomes 101: Basic Steps to Creating a Gnome

3.1 Introduction

The INFACT Gnome environment was designed for use by a variety of developers with backgrounds ranging from extensive development experience with the INFACT environment to basic familiarity with INFACT and a desire to make use of Gnomes as a solution to some problem. Concordantly, this chapter focuses on outlining the basic steps essential to creating and deploying an autonomous agent using the tools provided by the Gnomes environment. It is not all-inclusive, and should be considered only a starting point when jumping into the world of Gnome programming. For more information, consult the references listed at the end of this paper.

3.2 Gnome Developer Prerequisites

While often conceived and designed by domain experts such as educators, the actual implementation of Gnomes is completed by individuals comfortable with the task of software development. Basic prerequisites to creating a Gnome include:

1. At least a basic level of familiarity with the INFACT environment. Developers should be familiar with the tools provided by INFACT, in addition to the various types of data gathered by the system.

2. Fluency with the Java programming language. Familiarity with Java exceptions and events, in addition to experience with elementary data structures such as the Java `Hashtable` and `Vector` is especially important.

3. A user account on the *kirsch* server cluster, in addition to membership in the *infact* unix group.

4. Familiarity with the Gnome Environment. This document should be sufficient to satisfy this prerequisite.

Before beginning work on a new Gnome, developers must also add the following directories to their `PATH` and `CLASSPATH` directories. Note that these paths assume development is taking place

on the kirsch server. When developing on ole, all occurrences of "kirsch" in the paths listed below should be replaced with "ole."

To their `PATH` environment variable, users should add:

- `/kirsch1/GNOMEENV/tools`

To their `CLASSPATH` environment variable, users should add:

- `/kirsch1/GNOMEENV/WEB-INF/classes`
- `/kirsch1/GNOMEENV/WEB-INF/lib`

3.3 Gnome Environment Source Map

To aid users in navigating through the Gnome environment utilities and source code, this section consists of a map of the source code, library files and utilities included in the Gnome Environment. Please consult the online Javadoc and the INFACT developer wiki web for more detailed information.

3.3.1 Overview

At the top level, the Gnome Development Environment consists of three primary components: the Gnome Environment server, the Gnome Data Interface API, and the source directories for individual Gnomes.

3.3.2 The Gnome Environment Server

Installed as the Tomcat web application `GnomeEnvironment`, the Gnome Environment server contains the components of the Gnome Environment that manage and run Gnomes. A copy of the Gnome Environment Server source can be obtained by checking out the `GnomeEnvironment` CVS module.

- **GnomeEnvironment**
  - **logs**
    The logs directory contains the individual log files corresponding to each Gnome installed on the system. When user's call a Gnome's `log()` method, this is where the message is logged.
  - **notes**
    The notes directory contains a series of README files with information pertinent to the development of the Gnome Environment. Gnome developers typically have no use for the information in these files. However, if a developer should encounter a problem with the environment, there may information about it here.
  - **persistent_data**
    The persistent_data directory contains the files to which Gnomes' serialized persistentDataStore hashtables are written.
  - **registry**
    The registry directory contains the lockfiles used by the `register_gnome` and `unregister_gnome` scripts.

- **tools**
  The tools directory contains the shell scripts that comprise the Gnome Rapid Development Toolkit.
- **WEB-INF**
  The WEB-INF directory is a part of the directory web application directory structure convention enforced by the Tomcat server. It contains all of the Java source, .class and .jar files included in the Gnome Environment.
  * **web.xml**
    The web.xml file contains all of the Tomcat Web Application configuration information for the GnomeEnvironment web-app.
  * **classes**
    · **.java and .class files for installed Gnomes**
    · **BaseGnome.class**
    BaseGnome.class defines the class from which all individual Gnome classes descend.
    · **Dispatcher.class**
    Dispatcher.class contains the implementation of the Gnome Dispatcher.
    · **gnome_events**
    The gnome_events directory contains the implementation of the Gnome Events API.
    · **gnome_exceptions**
    The gnome_exceptions directory contains the .java and .class files for the various Java exceptions thrown by the Gnome Environment web application.
    · **gnome_fileutils**
    The gnome_fileutils directory contains various .java and .class files for easing the task of performing file operations withing Gnome implementations. This segment of the system is still under development, and its use is not currently supported.
  * **lib**
    The lib directory contains a series of .jar files that are implicitly referenced by all Gnomes. JAR files for the Gnome Data Interface, as well as JAR files created for individual Gnomes are stored here.

3.3.3 The Gnome Data Interface

The `GnomeDataInterface` directory contains the implementation of the Gnome Data Interface API. A copy of the Gnome Data Interface source can be obtained by checking out the `GnomeDataInterface` CVS module.

- **gnome_data_interface**
  The gnome_data_interface directory contains the actual source comprising the Gnome Data Interface API.
  - **DataInterface.java** and other supporting .java files
    DataInterface.java contains the implementation of the DataInterface object, the entry point when using the Gnome Data Interface. Other .java files in this directory support DataInterface.java.

  - **gconstants**
    The gconstants directory contains classes defining a number of constant values used throughout the Gnome Data Interface API.
  - **gdatatypes**
    The datatypes directory contains the definitions for the various datatypes used by the Gnome Data Interface.
    * **basic_types**
      The basic_types directory contains the generic datatypes used by the Gnome Data Interface.
    * **forum_types**
      The forum_types directory contains the INFACT-specific datatypes used by the Gnome Data Interface.
  - **gexceptions**
    The gexceptions directory contains the files defining the various exceptions thrown by the Gnome Data Interface.
  - **gfetchers**
    The gfetchers directory contains the classes that define the various Fetcher objects included in the Gnome Data Interface.
  - **gputters**
    The gputters directory contains the classes that define the various Putter objects included in the Gnome Data Interface.
  - **gschema**
    The gschema directory contains classes that encapsulate information on the schema of INFACT databases.
  - **gutils**
    The gutils directory contains classes that facilitate low-level interaction with SQL databases.

- **lib**
  The lib directory contains JAR files, such as the POSTGRESQL driver, used to support the Gnome Data Interface.

- **tests**
  The tests directory contains unit tests for the various components of the Gnome Data Interface.

3.3.4 Individual Gnome Source Directories

Individual Gnome Source directories are generated automatically by the `make_gnome` script, and contain individual Gnomes' implementation files, supporting .jar and .class files, and a Makefile.

- **build.xml**
  build.xml contains resource information that allows the individual Gnome's Makefile to restart the Gnome Environment when installing the Gnome on the server.

- **Makefile**
  Makefile contains *all*, *install*, and *clean* targets that aid in the compilation of Gnomes.

- *GnomeName*.**class**
  GnomeName.class contains the actual implementation of the Gnome.

- *GnomeName*
  The *Gnome Name* directory is the location where all .class and .jar files supporting an individual Gnome should be stored. When the Gnome is compiled and installed, the files in this directory are archived in a .jar file and made available to the Gnome at run-time.

3.4 Creating and Managing the Source for a New Gnome

3.4.1 Generating the Source Tree

The first step to creating a Gnome is the generation of a Gnome source tree and boilerplate implementation file. Assuming that the user's `PATH` and `CLASSPATH` environment variables are configured properly, a new Gnome source tree can be created by issuing the following command:

make_gnome *gnome name*

3.4.2 Compiling and Installing Gnomes

Once the user has created a source tree for their new Gnome, they are free to begin implementation immediately. When ready to compile their new Gnome, the user can do so by issuing the command from within the top-level of the source tree:

make

This command compiles the Gnome within the local directory, but does not install the it. Once the Gnome has been compiled, installation can be accomplished by issuing the command:

make install

Running `make` on the *install* target copies the Gnome source to the server distribution directory, compiles it, re-starts the Gnome Environment and wakes up all Gnomes registered with the system. Once this process has completed successfully, the new Gnome should be installed on the server and running.

3.4.3 Uninstalling Gnomes

To remove a Gnome, the user should issue the command:

remove_gnome *gnome name*

Once make has completed, users should then remove the source tree corresponding to the removed Gnome from their local directory.

3.5 Essential Methods to Implement

Providing a Gnome with its desired functionality is accomplished by implementing the methods within the main class defining the gnome that correspond to the various modes of Gnome execution. These methods, along with a brief description of their intended usage are listed below.

3.5.1 `startupGnome()`

The `startupGnome()` method is called one time as the server comes up. This method gives users a chance to retrieve saved information from disk or a database, and to initialize that state of a Gnome when first started.

3.5.2 `subscribeToEvents()`

The `subscribeToEvents()` method is called one time as the server comes up, in the same manner as `startupGnome()`. This method should only be used for the purpose of subscribing to Gnome Events. Please see the section below on utilizing the Gnome Event system.

3.5.3 `shutdownGnome()`

The `shutdownGnome()` method is called one time as the server shuts down. It is typically used for tasks such as releasing resources like open database connections and file handles, or for saving the state of a Gnome before the server goes down.

3.5.4 `handleGnomeEvent(GnomeEvent event)`

The `handleGnomeEvent(GnomeEvent event)` method is called each time an event of a type subscribed to by the Gnome occurs. The `event` parameter contains the Gnome event object itself. It is important to note that the `handleGnomeEvent` method can be called any number of times, often in parallel. As a result, class member variables accessed by `handleGnomeEvent` need to be synchronized.

3.5.5 `batchExecute()`

The `batchExecute()` method is called when the Gnome is run in batch execution mode. Similar to `handleGnomeEvent` it can also be called multiple times, possibly in parallel.

3.5.6 `immediateExecute()`

The `immediateExecute()` method is called when immediate execution of the Gnome is requested through the Gnome Web Interface. This method can also be called multiple times, possibly in parallel.

3.5.7 `main()`

The `main()` method is used primarily for the purpose of debugging Gnomes. `main()` is called when Gnomes are invoked from the command line, and functions in the same way as the `main()` methods of other Java applications.

3.6 Making Use of Gnome Events

The Gnome Events system provides a great deal of versatility to Gnomes, allowing them to respond in real-time to actions taking place within INFACT. Implementing a Gnome that takes advantage of the Gnome Events system requires two steps: subscribing to the event types relevant to the Gnome and implementing a listener to process incoming events.

To make use of the Gnome Events system, Gnomes must import the `gnome_events` and `gnome_events.exceptions` package.

3.6.1 Subscribing to Gnome Events

Client Gnomes subscribe to events within the `subscribeToEvents()` method. This is accomplished by making calls to the `subscribeToGnomeEvent` method, which takes a Gnome Event type identification code as a parameter. Constants defining the valid event types can be found in the `gnome_events.GnomeEventTypes` class. Please refer to the `gnome_events` Javadoc files for a more detailed explanation of the available event codes.

Once the appropriate subscription request statements are included in a Gnome's `subscribeToGnomeEvents` method, it should begin receiving events as soon as the Gnome is installed.

3.6.2 Processing Gnome Events

Gnome Event reception and processing is handled by the `HandleGnomeEvent` method within the class defining a Gnome. The event itself is passed to the `handleGnomeEvent` method through the `event` parameter of type `GnomeEvent`. While `GnomeEvent` itself is abstract, all event types supported by the system descend from it.

A common strategy for processing Gnome Events is to place a series of `if` statements using the `instanceof` operator to determine the type of the incoming event, forking to private methods defined to handle events of only a single type. It is worthwhile to note that the events for all forums linked to the Gnomes environment are dispatched to all Gnomes subscribed to that particular event type, so it is often useful to filter events using the `database` field, which contains the name of the forum that generated the event.

3.7 Putting the Gnome Data Interface to Work

The Gnome Data Interface provides Gnomes with a link to all of the information within the INFACT system, in addition to a link with the outside world through e-mail. Users can take full advantage of the GDI by following the steps outlined below:

3.7.1 Using the DataInterface Object

To use the Gnome Data Interface, users must first import the `gnome_data_interface` packages within the source files for their Gnome. These lines are already included (but commented out) in the source files generated by the `make_gnome` script.

With the proper libraries included, the first step to using the GDI is creating a `DataInterface` object by calling it's default constructor. As a rule of thumb, one `DataInterface` object is typically shared between all the methods of a Gnome class. The `DataInterface` object is then instantiated within the `startupGnome()` method.

3.7.2 Establishing a Persistent Database Connection

Once a `DataInterface` object has been created, the user must establish the connections with one or more INFACT databases before the core functionality of the GDI can be used. This is accomplished by making calls to the `DataInterface` object's `establishDatabaseConnection` method, which takes the name of the database with which to connect as a parameter. As GDI database connections persist indefinitely, they are typically established within the body of a Gnome's `startupGnome()` method immediately after the `DataInterface` object is instantiated.

Persistent database connections are automatically destroyed when Gnome classes are deallocated as the Tomcat server shuts down. Users can explicitly destroy database connections by using the `DataInterface` object's `destroyDatabaseConnection` method, which takes the name of the database connection to be destroyed as its only parameter.

3.7.3 Using Getters and Fetchers

With a `DataInterface` object instantiated and database connections established, Gnomes are then ready to begin creating Getter and Fetcher objects through which forum content is actually obtained or created. Unlike `DataInterface` objects and database connections, which persist from server startup to shutdown, Getters and Fetchers are designed to span a much shorter lifetime. They are typically allocated, put to use, and discarded.

Getter and Fetcher objects are obtained by making calls to various Getter/Fetcher creation methods within the `DataInterface` class, all of which take the name of a valid database connection as a parameter. For more information on using Fetcher and Getters, please consult Section 2.6 or the Gnome Data Interface Javadoc.

3.8 Using Auxiliary Source Files and Libraries

When authoring Gnomes, users may wish to make use of auxiliary Java class files and JAR archives. The Gnome Environment provides a mechanism by which code within Gnome definition classes can easily leverage such external resources.

When a new Gnome is created, the Gnome Source tree contains a subdirectory with the same name as the Gnome. To make additional class and JAR files available to Gnomes, the developer need only place the relevant files in this directory. When the developer runs `make install` the resource files will be installed to the server and made available to the Gnome at runtime.

3.9 Hints and Strategies for Developing Gnomes

The following list of general hints and strategies has been compiled to aid new developers in the task of developing new Gnomes:

- Work in a top down manner. Start by sketching out the desired functionality and structure of your new Gnome, writing empty methods before diving into the details.
- Work in small steps, testing along the way. The complex environment in which Gnomes execute has many quirks and pitfalls. To keep on the right track when developing a Gnome, avoid making assumptions. Take as small of steps as possible in between tests.

- Make extensive use of the Gnome `log()` method. The nature of the Gnomes system makes the use of a debugger such as jdb impossible. Writing messages to the server error log can help immensely in developing and debugging Gnomes.
- If a Gnome doesn't seem to be executing, be sure that you have run `make install`. Restarting the server (which the install target of Gnomes Makefiles accomplishes) is necessary when installing or updating Gnomes.
- Keep the Javadoc for the Gnomes Environment and the Gnome Data Interface handy.
- Explore the code written by other Gnome developers. The best way to learn the ropes of Gnomes programming is to examine the Gnomes written by others. The Java source files for Gnomes installed on the system can be found in the `/WEB-INF/classes` directory within the Gnome Environment.

Chapter 4

Gnomes Currently in Action

In an effort to prove the concept of the INFACT Gnome Environment, a number of simple Gnomes have been developed. This chapter describes two such Gnomes, one already deployed, and the other currently in development.

4.1 The Auto-Feedback Gnome

4.1.1 Overview

The size of introductory college courses often prohibits teachers from providing rapid feedback to student responses. In many cases, however, even very general feedback such as confirmation as to whether or not a student is on the right track can be useful.

With this idea in mind, we have designed, implemented and deployed an "Automatic-Feedback Gnome"- an agent that waits listening for student posts to the INFACT-FORUM, performs some basic pattern matching on the text posted by students, and responds by sending e-mail messages based on the content of the examined forum posts.

4.1.2 Rules and the INFACT-RULEBASE-EDITOR

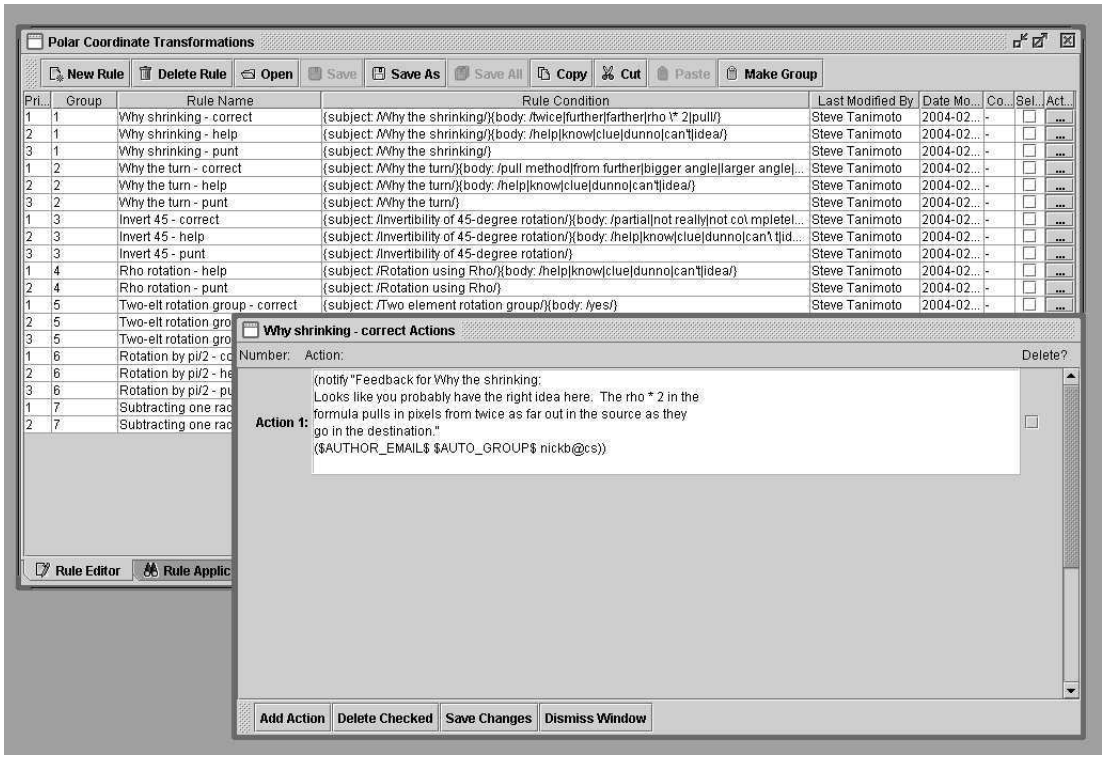
The Automatic Feedback Gnome makes use of INFACT Rule-bases to facilitate the analysis of INFACT-FORUM posts. INFACT Rule-bases are composed of one or more *rule groups*: sets of text-matching rules related to the same fine-grain concept.

Rules consist of two primary components: a rule condition, used to find text indicative of a conception held by a student, and a rule action which specifies the steps to be taken when the rule condition finds a match. Rule conditions are essentially regular expressions with a few modifications. In addition to basic regular expression functionality, rule conditions also allow users to specify ordering and distance constraints on the words comprising the text to be matched. The action component of the rule specifies the measures to be taken when the pattern component of the rule finds a hit within the text being searched. Represented using a custom scripting language, typical rule actions consist of sending an automatically generated e-mail message, posting to the INFACT-FORUM or generating INFACT-MARKUP data.

In addition to condition and action components, rules also possess an execution priority rank. When the application of a rule-base takes place, execution of only a single rule within each rule group is allowed. When the system applies a rule-group, it begins applying rules from highest to lowest priority until a rule within the group matches. As a result, users can specify the order in

which rule application takes place within rule groups by modifying priority ranks. A typical strategy employed by rule-base developers is to assign priorities corresponding to the relative specificity of rule conditions within a group. In this manner, the system first attempts to make specific inferences based on detailed patterns, then resorts to making more general inferences if specific conditions fail to trigger a match.

Rule-bases are created and managed using the INFACT Rulebase Editor, a Java applet developed during Summer 2003. The Rulebase Editor tool provides rule-base developers with an easy means for entering, organizing and debugging INFACT rule-bases. The system provides tools for automatically checking the syntax of rule condition and action specifications in addition to a "rule applicator" tool that allows users to apply rules to pre-existing INFACT-FORUM posts, viewing the matches in real-time.



The INFACT-RULEBASE Editor

4.1.3 How the System Functions

The Auto-Feedback Gnome provides system for automatically applying rule-bases to INFACT-FORUM posts as they are made. The Gnome is implemented using the event-based Gnome execution paradigm.

Subscribing to INFACT-FORUM post events, the Auto-Feedback Gnome waits for student responses to questions posed by course instructors. When a new post event is received, the Gnome takes the following steps:

1. The Gnome consults a database table specifying which rule-bases should be applied to which INFACT-FORUM groups determining which, if any, rule-bases should be applied to the incoming post.

2. The appropriate rule-bases are applied to the post.

3. In the case where a rule condition matches a segment of the forum post, the gnome parses and executes the action associated with the rule, making use of the Gnome Data Interface to carry out the tasks specified within the action.

4.1.4 Use within GIS 130: Pixels, Numbers and Programs

The Auto-Feedback Gnome was successfully used within GIS 103, an introductory course in image processing offered at the University of Washington during Winter Quarter 2004. While unable to provide as detailed or reliable of feedback as that generated manually by course staff, the system was often capable of providing feedback that gave students a hint steering them in the right direction.

It is hoped that through refinement of the rule-bases used for Pixels, Numbers and Programs in addition to improvements in the rule condition language the Auto-Feedback Gnome will be even more effective in future offerings of this particular course.

4.2 Bayesian Classification Gnome

4.2.1 Overview

When using the INFACT-MARKUP tool, manually selecting a facet to associate with text highlights can often be a very tedious task, especially when a large number of posts are being examined. Facet association, the process by which a users browses through a catalog of facets selecting the most relevant entry to associate with a segment of text, is essentially a text-classification process. The Bayesian Classification Gnome attempts to automate at least a part of this process by making use of a Naïve Bayes text classifier.

4.2.2 Naïve Bayesian Classification and Facet Associations

Recall Bayes' Rule:

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)} \tag{4.1}$$

In the case of attempting to classify text in terms of its relevance to a particular facet, the above equation can be written as:

$$P(facet|text) = \frac{P(text|facet)}{P(text)} \tag{4.2}$$

For the Bayesian Classification Gnome, we use word occurrence as the feature upon which to base text classification, so:

$$P(text) = P(word_1 \wedge word_2 ... \wedge word_n) \tag{4.3}$$

To simplify the calculation of text string probabilities, we make the so-called Naïve Bayes assumption, assuming that the probability of each word is independent of all others in the dataset. When assuming variable independence,

$$P(a \wedge b) = P(a)P(b) \tag{4.4}$$

so the probability of a given text string can be calculated as:

$$P(text) = P(word_1) \cdot P(word_2) ... \cdot P(word_n) \tag{4.5}$$

$P(text|facet)$ is obtained by calculating $P(text)$ taking into account only the words within the examples already manually classified as $facet$. $P(facet)$ is obtained by dividing the number of examples manually classified as $facet$ by the total number of manually classified examples.

Once the above probabilities have been calculated for an observed text string, we can apply Bayes' Rule and classify the string as being associated with the facet within the catalog for which $P(facet|text)$ is maximal.

4.2.3 How the System Functions

The Bayesian Classification Gnome operates in two distinct modes: training and classification. In training mode, the Gnome subscribes to facet-association events, building up a word probability table (stored using the Gnome Persistent Data-store) based on the strings involved in the facet associations.

In classification mode, the Gnome also subscribes to facet-association events, however, upon reception of a new facet-association event, the Gnome takes the following steps:

1. The probability of the observed text string being associated with each facet in the catalog is calculated using the probability tables in the Persistent Data Store
2. The text is classified as being related to the facet for which $P(facet|text)$ is maximal.
3. The Gnome Data Interface is used to associate the facet with the newly created assessment.
4. Using batch-execution mode, the Gnome e-mails reports on its activity to users whose markup activity triggered automatic facet associations.

Chapter 5

Future Work

The Gnome Environment in its current state, while proving useful in a number of applications, still only scratches the surface of the benefits assessment agents could provide within INFACT. This chapter outlines a number of extensions and improvements to the Gnome Environment that may be worth exploring in the near future.

5.1 New Gnomes

One of the most important challenges is the creation of new Gnomes. This section briefly touches on a number of possible Gnomes whose implementation has been discussed.

5.1.1 Teacher-Inspired Heuristic-Based Gnomes

Up to this point, considerable effort has been expended on the task of manually analyzing the app-event sequences generated by the INFACT-SKETCH tool and the Pixelmath calculator. In the course of this analysis, a number of heuristics have been identified when observing domain-experts in process of picking apart event logs.

A number of potentially useful event-log analyzing Gnomes could easily be based on these heuristics. Should this task be pursued, it will involve two primary steps: conducting detailed observation of various domain-experts analyzing event data, identifying and extracting heuristics and implementing a series of Gnomes based on such heuristics.

5.1.2 Porting Other Research Projects to the Gnome Environment

A number of projects relating to automated student assessment are currently being conducted within the Online Learning Environments Lab at the University of Washington. These projects could benefit greatly from being ported to run as assessment agents within the Gnomes Environment.

Projects that could be ported to run as Gnomes with a relatively small amount of effort include software that attempts to infer student thought processes from data within Pixelmath event logs using Hidden Markov Models. Another such project learns text-matching rules similar to those used within INFACT rule-bases by examining manual markup activity[5].

5.1.3 Bayes Nets

A system making use of Bayesian networks in a manner similar to that of rule-bases is currently in the initial stages of development. In the new system, users will construct INFACT domain-specific Bayesian Networks, or *graphical assessment models* that relate observable student actions with facets.

Once appropriate graphical assessment models have been constructed, a Gnome would be ideal for capturing observations using the Gnome Event system, updating the probabilities of the nodes within the model and generating some sort of feedback using the Gnome Data Interface.

5.2 Expansion of the Gnome Data Interface

Over the course of the past several months, the Gnome Data Interface has proved itself as a very useful tool for accessing the INFACT database, even in Java applications not explicitly related to the Gnomes environment. In the future, the Gnome Data Interface could potentially be expanded to replace the Java Database API currently employed by the various Java applets within the INFACT environment.

Advantages to using the Gnome Data Interface over the current API include:

- Increased performance due to fewer levels of indirection between the API and the database
- Tighter integration with Java Applets
- A simpler interface requiring no knowledge of the internal INFACT database layout
- Ease of expansion – the Gnome Data Interface can easily be modified to support changes to the internal database schema

5.3 Interactive Gnomes

As Gnomes grow more and more complex, a key element missing from the Gnome Environment is the ability for Gnomes to communicate directly with Java applets residing on client machines. Expansion of the Gnome Environment to facilitate such communication enables the creation of more intelligent Java applets within the INFACT system, backed by assessment agents residing on the server.

5.4 Enhancements to Environment Security and Performance

Up to this point, the experimental nature of the Gnome Environment has emphasized functionality over security and scalability. As the user-base of the Gnomes Environment grows, these issues will become more pressing. Needed improvements to the system may include:

- More robust security measures within the Gnome Event system to prevent Gnome event "spoofing."
- A "Gnomes Permissions" system allowing administrators to specify which Gnome Environment resources individual Gnomes are permitted to use.
- A more robust system for handling concurrent Gnome installation.

- The ability to distribute Gnome execution over multiple servers. The J2EE API provides this functionality, however, it was not explored as part of this project.

5.5 Support for Other Programming Languages

In the future, developers may wish to implement Gnomes in languages other than Java. Languages such as LISP and Python often prove more useful for tasks such as text analysis.

One possible means of providing support for Gnomes written in other languages is through the use of web services and the SOAP protocol. Under such a scheme, "multi-lingual" Gnomes would be provided with stubs enabling them to communicate with the Gnome Environment through XML-based remote procedure calls.

Gnome Developer References

Gnome Data Interface Javadoc: `http://kirsch.cs.washington.edu/~nickb/gdi`

Gnome Environment Javadoc: `http://kirsch.cs.washington.edu/~nickb/gnomes`

The OLE Lab Wiki-Web: `http://kirsch.cs.washington.edu/cgi-bin/BaseQuicki/wiki.cgi`

Acknowledgments
The author would like to thank Steve Tanimoto and Nathan Evans for providing invaluable feedback throughout the development of the system, Tim Whitcomb for proof-reading this document and Scott Batura for forcing me out of the lab from time to time.

Bibliography

[1] Minstrell, J. 1992. Facets of students' knowledge and relevant instruction. In Duit, R., Goldberg, F., and Niedderer, H. (eds.), Research in Physics Learning: Theoretical Issues and Empirical Studies. Kiel, Germany: Kiel University, Institute for Science Education.

[2] Tanimoto, S. L., Carlson, A., Hunt, E., Madigan, D., and Minstrell, J. 2000. Computer support for unobtrusive assessment of conceptual knowledge as evidenced by newsgroup postings.*Proc. ED-MEDIA 2000*, Montreal, Canada, June.

[3] Tanimoto, S. Exploring mathematics with image processing. *Proceedings of the 1995 IFIP World Conference on Computers in Education*, Birmingham, UK, July 23-28. London: Chapman & Hall, (1995) 805–814.

[4] Byrne, C. Edwards, P. 1994. Building Agent-Based Systems. *University of Aberdeen Computing Science Technical Report AUCS/TR9405*

[5] Carlson, A. Tanimoto, S. 2003. Learning to Identify Student Preconceptions from Text.*HLT/NAACL 2003 Workshop*, Edmonton, Canada.

[6] Hunter, J. 2001. *Java Servlet Programming* (2nd ed.). Sebastopol, CA: O'Reilly.