Information Engineering Technology

GuardIEn

Introduction

GuardIEn is a configuration management system that supports and automates the control of Gen models.

Providing comprehensive change control and model management facilities, it is a software product for organisations who wish to provide effective control over applications developed with Gen, and who need to reduce the overhead required to implement these controls.

GuardIEn provides the following key functions:

- **Version control**, with status tracking at the object level using project specific life-cycles and detailed tracking of changes at the object property/statement level.
- Integrated **change control** facilities.
- Sophisticated **impact analysis** tools which access data directly from the encyclopaedia, allowing rapid impact analysis, ‘where used’ and ‘when changed’ reporting.
- Automated **object migrations**.
- **Automated updating** of controlled environments (for example system test or production) including object migration, impact analysis, source code generation and installation and other tasks necessary to create and maintain an executable system.
- **Interfaces** to enterprise configuration management tools like Endevor, Harvest and ChangeMan.
- Integrated **code review** capabilities.
- **Web Interface** for Change and Release Management

Background

The successful management of software development projects has always required a high degree of control over the source code. Up until the use of model based development tools, the controllable item was program source code. Library management utilities were required to control changes to source code: to ensure that changes were accurately installed in the production system and to provide an audit trail.

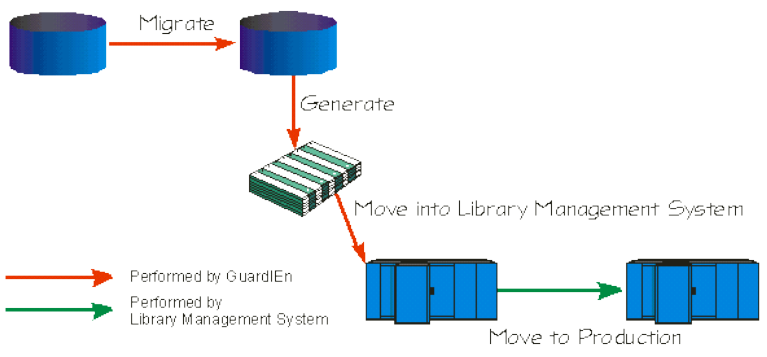
With Gen, the true 'source' of the system is no longer the generated code, but the model(s) from which the code is generated. Changes to the application are applied to the Gen models, and therefore controls should be applied to the contents of the encyclopaedia and not just to the program source code.

The use of Gen does not remove the need to control deliverables. Indeed, the greater speed and responsiveness of a Gen based development environment coupled with greater sharing and reusability of components increases the need for well-defined and supported configuration management procedures. Many organisations will wish to retain their existing library management tools for managing the implementation of code into production and therefore have a requirement to integrate Gen with these tools.

Attempts to directly interface Gen with source code based library management tools rarely achieve the desired results because these tools have not been architected to work with Gen. They cannot issue migrations, execute the code generators or perform impact analysis on the contents of the model. The handover of control to the library management system is therefore performed once the source code has been generated and installed. At this point, however, most of the effort will have been expended performing the Gen steps. Furthermore, errors in the implementation are normally caused by forgetting to migrate, generate or install the necessary components and the resulting implementation will therefore contain these errors. In contrast, GuardIEn has been designed to work with the Gen encyclopaedia. It understands how to version control Gen objects, migrate between models, automate impact analysis, execute the Gen code generators and compile/link/bind the code. The result is that GuardIEn ensures error free implementations.

But GuardIEn does not just control the movement of code into production. It also offers a high degree of support during the development stages of a project. It can automate model management for a wide variety of model management approaches, offers sophisticated impact analysis and reporting capabilities and can also automate the code generation and installation steps for the development environments.

GuardIEn can also work with existing configuration management tools like Endevor, ChangeMan and ISPW so that these can be retained to perform the final stages in a production implementation. The figure below illustrates the integration of Gen and a library management product using GuardIEn to provide the interface.


Migrate
Generate
Move into Library Management System
Move to Production
Performed by GuardIEn
Performed by Library Management System

Change Management

Without a clear definition of the changes that have been applied to the models, it is almost impossible to perform an incremental update to the target system, resulting in the need to regenerate and re-implement the entire application.

The clear specification and management of changes to the application is therefore an essential prerequisite to the efficient and rapid implementation of the changes and also to automating the implementation processes.

There are many different approaches that can be adopted to defining changes, from paper-based systems to automated databases. However, these often fail to address the key issue with change control, which is that the change requests need to be linked to the deliverables that were created or modified during the implementation of the change.

GuardIEn contains a change control module that enables the central definition of change requests. It is integrated with the other GuardIEn modules, and includes the ability to automatically associate the change requests to affected deliverable versions.

Change Requests can be grouped into a GuardIEn Release Pack. This provides the scope and authorisation for the automated implementation into a target environment using the GuardIEn system updating module, and ensures that related changes are implemented together.

The integrated Upload Assistant will automatically scope new, changed or deleted objects into the Change Request, removing the need for the developer to manually record the changes that they have made to the model.

Version Control

The concept of versioning deliverables is key to the effective automation of configuration management, but it also requires the ability to explicitly and uniquely identify individual versions. Whilst Gen allows multiple versions to be stored, it does not allow them to be explicitly identified. GuardIEn complements Gen by enabling the explicit identification of versions. The data about the versions is stored in the GuardIEn database and linked to the underlying objects in the models.

Each deliverable is individually versioned in GuardIEn. This enables the tracking of the versions that have been created for a deliverable, the release of the system they are included in, and their progress through the development life-cycle. Versions are automatically created by GuardIEn when changes are uploaded to the encyclopaedia using the integrated Upload Assistant.

GuardIEn can be setup to record each change made to an object, when it was changed and which userid applied the change. It can store the action diagram (PAD) statements for each change and produce a comparison between any two versions. This allows statement level tracking of changes to an action diagram without requiring separate models.

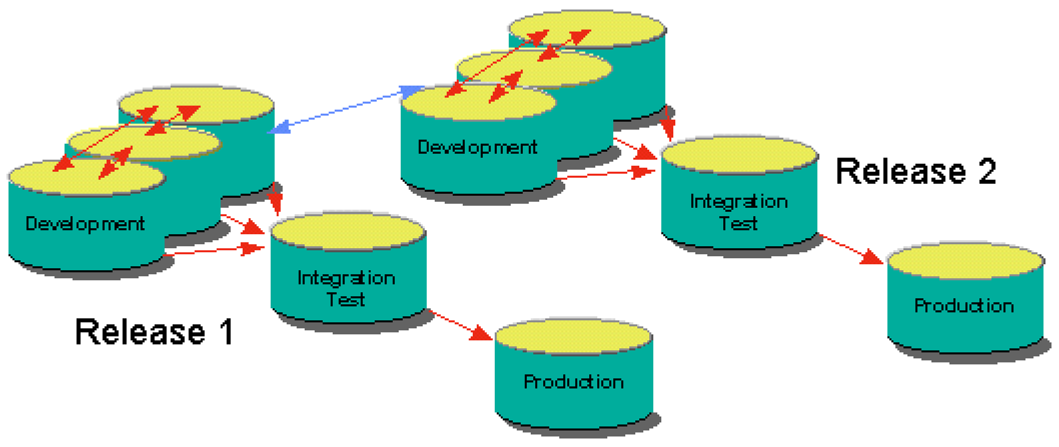
Since GuardIEn stores the data for the minor versions in its own database, it is able to provide statement by statement, property by property tracking of changes. This is not possible if you only rely on the Gen encyclopaedia since a Gen model only contains the latest version of an object.

Release Management

An important part of effective configuration management is a clear and well understood release management policy. This helps plan when changes are to be applied to the production system. It should cater for both the medium to long-term enhancements as well as short term fixes that may be required.

GuardIEn helps projects to define and plan release implementations by providing facilities to document the releases and their content. Multiple system releases can be defined as either independent releases, or linked as a hierarchy of releases which facilitates the tracking of changes between releases.

GuardIEn allows the parallel development of system releases. This enables the team to work on more than one release at the same time. The diagram below illustrates an example model architecture for parallel development.



One of the problems with parallel development is ensuring that any changes or fixes that are applied to a release are also applied to the subsequent release. This is illustrated in the diagram above by the blue double headed arrow linking the development models for releases 1 and 2.

GuardIEn contains special facilities for tracking changes in one release and detecting whether they can be safely migrated to the next release or not. It uses a concept called 'baselining' to detect whether the object has been changed in the subsequent release and thus whether it can be migrated or whether the change has to be manually applied to the next release to avoid losing any changes made in that release.

Impact Analysis

GuardIEn contains facilities to assist in rapid and effective impact analysis. It directly accesses the contents of the models to provide answers to questions like:

- When was an object last changed in each of the models that contain it?
- When was it last generated?
- When was it last migrated?
- What objects use the object, or what objects does it use?
- What objects have been changed but not regenerated?
- Is the current model timestamp the same as the last time it was placed into production?
- What subsets is the object checked out to?
- What change requests have scoped the object?

All of these questions and more can be answered very quickly by using GuardIEn's integrated impact analysis features. Because the data is directly extracted out of the encyclopaedia, it is up to date, and unlike workstations reports that operate on a subset of the data, it is complete because it is extracted from the complete model.

GuardIEn's reports go beyond single model reporting - they can span multiple models, providing the sort of powerful cross model reports that are required to manage a multiple model development environment.

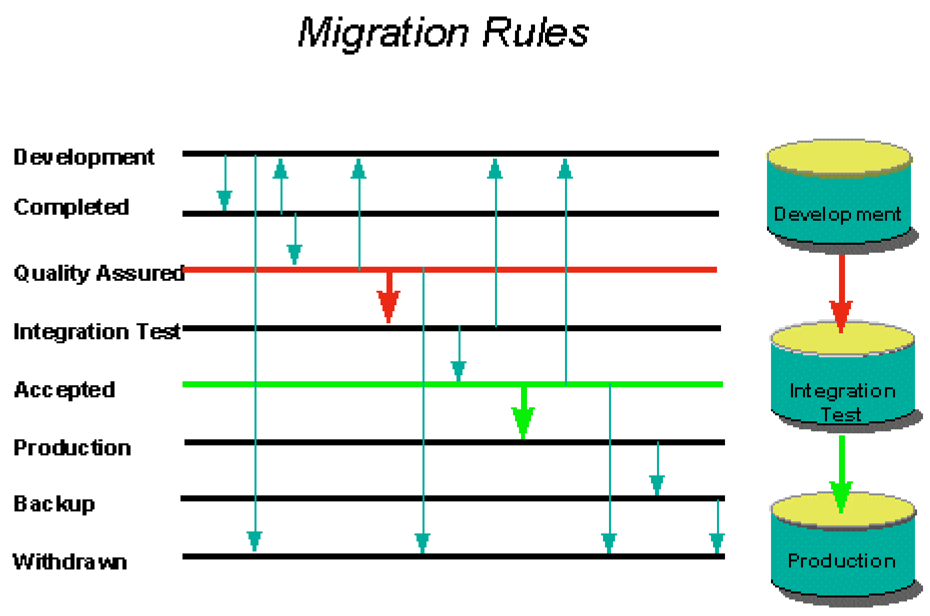
Model Management

GuardIEn automates many of the checks and tasks that are necessary to successfully manage multiple Gen models. It allows the project to define its model architecture using a rules based approach. The migration rules specify the points in the deliverable life-cycle that indicate that a migration is required, the preconditions and post-conditions for the migration and which model or models are to be targeted.

To ensure that the migration is performed from the correct model for each deliverable, GuardIEn allows the nomination of a 'master' model for each deliverable. This indicates the model where changes are to be initially applied, and hence which model to use to update the other development models when a new version is to be migrated. This greatly reduces the risk that the production system is updated from the wrong model, or that the master version of the deliverable is accidentally overwritten by migration from the wrong model.

GuardIEn also performs 'enabling' object checking, automatically adding required objects into the migration. This greatly reduces the number of migration failures since missing enabling objects are the most common reason why a migrate does not succeed

The diagram below illustrates the linking of two status changes in the life-cycle to corresponding migration paths in the model architecture.



GuardIEn's migration rules provide great flexibility in enabling the product to be able to address most model architectures.

Automated System Updating

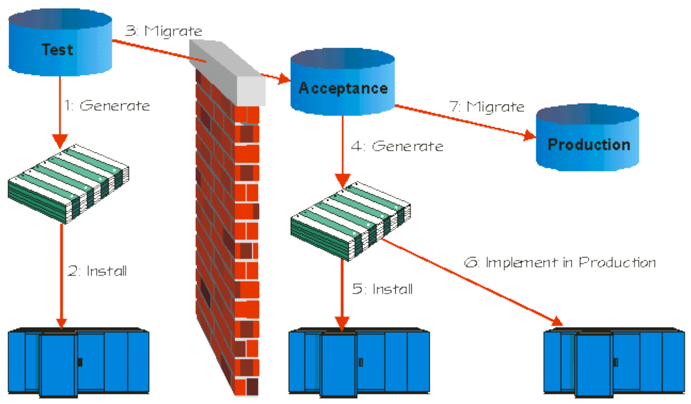
GuardIEn automates the updating and implementation of changed objects into a target environment, including the updating of the executable system.

The steps required to implement a system will vary between organisations. Typical steps include:

- Migrate changed deliverables to a backup model.
- Migrate new and changed deliverables to controlled model.
- Regenerate RI triggers.
- Compile external action blocks.
- Execute Gen code generators to generate source code.
- Install source code (compile, link-edit and optionally bind).
- Interface to library management or production updating systems.

GuardIEn executes the steps using a series of linked background tasks. It executes each step, checks the results and if the step was successful, automatically moves on to the next step. This allows the update to be performed overnight without any intervention. Contrast this with a manual process whereby each step has to be performed individually, probably split over several days to allow for example the completion of one step before defining and executing the next step.

The figure below provides an example of how the GuardIEn system updating process can automate some typical steps in a production update. It can first update the test libraries. Once the testing is complete, the objects are migrated to a controlled acceptance model. The acceptance model can be secured so that only GuardIEn has access to it, signified in the figure below by the brick wall. The code is then regenerated and installed into acceptance to guarantee that the model is synchronised with the source and object code. Once acceptance testing is complete, the changes can be implemented into production, either by GuardIEn or by existing production updating routines invoked by GuardIEn. This is followed by a migrate to the production model to synchronise the production model with the production code.



Benefits

- ✓ **Eliminates system errors** attributable to poor configuration management by automating the object migration, impact analysis, code generation and system updating processes. Timestamp errors and other system failures are avoided.
- ✓ Significantly **reduces the effort** required to manage a Gen development environment.
- ✓ Enables changes to be **implemented much faster** through the use of the automated system updating routines that perform all of the required implementation steps in a series of linked background tasks.
- ✓ Provides **greater control over changes** by linking changed objects to the originating change requests, ensuring that only authorised changes are implemented into production.
- ✓ Provides **rapid analysis of the impact of changes** to the system, and allows the result of this impact analysis to be directly used as input to the scope of a change request.
- ✓ Allows all relevant individuals to **track the status** of a change request from it being raised to being completed. This allows accurate and timely monitoring of progress, reducing the need to 'chase' the status of change requests and the consequent risk that a set of changes are mislaid or forgotten.
- ✓ The product's open architecture and customisable exit routines enables **integration with enterprise change management** tools.