Generic Adapter Logging Toolkit

Genady Grabarnik  Abdi Salahshour  Balan Subramanian  Sheng Ma
IBM T.J. Watson Research Center
19 Skyline Dr., Hawthorne, NY 10510, USA
genady,abdi,balan,shengma@us.ibm.com

Abstract

The Generic Adapter for Autonomic Computing provides a framework for a unified approach to transform software messages and events into a standard situational event format in the autonomic computing architecture. The Adapter as an application supports self-configuration and self-optimization. This is a poster version of paper published elsewhere[5].

1. Introduction

We build computing systems (solutions) to satisfy the desires of the owners and users of the systems. Complexity of such systems increases and, hence, they are progressively more costly to install and to maintain. System administrators must deal with hundreds of subsystems and thousands of parameters just to keep systems running. Administrative cost far outstrip hardware and software acquisition costs. We are reaching the limits of complexity in the systems we build. Resolution of the situation is the goal of Autonomic Computing (AC). It aimed to provide solutions with dramatically decreased requirement for human administration in comparison to currently existing systems (see for example [1],[2]).

1.1. Use Case Scenario

The value proposition of Generic Adapter can be better realized by walking through a scenario. Consider a case where a distributed solution consists of heterogeneous components, for example, WebSphere Application Server (WAS), DB2 UDB, MQ Series, and some other non-IBM software deployed in a network with multiple routers and servers. Each of these components is well equipped with logging and tracing facilities reporting on it’s operating status and situation. However, the message logs are invariably "product-centric" and reflect standards and terminology that are unique to a particular vendor or even application. Different format of the data makes it very difficult to filter, correlate, and analyze messages across solution components. Therefore, a mechanism is required to collect data as accurately as possible to allow analysis engines and autonomic management to make appropriate decisions and to plan independently from original data format. To address this problem, let us introduce the Generic Adapter, a smart rule-based data collector. The Adapter is capable of reading the product log files (or directly receive messages), and convert them into a unified format (i.e., Common Base Event) that identifies the reporting and source component(s) as well as source situation.

2. Architecture and Implementation

Overall architecture of the toolkit includes three major types of components: run-time translator, repository of rules, Rule Builder. Each type carries out the following functionality: run-time translator provides translation of the messages to the unified format, repository serves as a storage for translation rules and best translation practices, and Rule Builder offers a console to allow configuration of the application as well as facilitation in creation of parsing rules. Each Generic Adapter component type relates to AC twofold: they are parts of the log messages unification process and they support self-configuration and self-optimization. The subsequent sections will discuss in more details main design points and supporting architecture for the run-time client as well as some specifics of Rule Builder component.

2.1. Main Design Points

- Provides generic parsing capability for handling different event/log entry formats in an adaptable way. Re-
ceives, converts and merges data in different formats from data sources into a canonical data format.

- Supports extensibility of the components
- Supports controllability and dynamic configuration
- Intelligently facilitates parsing rules creation and supports best practices

2.2. Resulting Architecture

Adapter Framework Layer manages the overall data flow and coordinates all components. It provides controllability and management, a control mechanism for lifetime/change request management, configuration changes and dynamic parser and filtering rule changes, dynamic adapter configuration in supporting active data collection and just-in time data collection, a support for pluggable architecture to extend main functionalities. It supports basic infrastructure and management functions including logging (e.g., saving incoming data for future usage) and data persistence support (e.g., handling data burst and possible connection cut-off), regular expressions optimization, and generic component/object initialization mechanism. It also supports communication and coordination with Generic adapter management processes. The implementation uses Component component as well as Context component. Each component of Generic Adapter extends Component class. A handling in the Generic Adapter separated on processing lines by log producers. The specific components (Sensor, Message Extractor, Parser Engine, etc.) in one processing line are contained in the Context specific for this line. Context also contains common variables for the processing.

2.3. Consolidation

Consolidation is one of the essential functions of Generic Adapter. It is represented by different filters, which, in their turn are separated on one message based filters (StringFilter, AttributeFilter, AggregateFilter) and filters based on group of messages (PatternFilter, ThrottleFilter, etc.).

2.4. Rule Builder - case base rule generation

Rule Builder as the authoring aid hides the xml schema and possibly complex rule expressions. This tool supports usage of best practices in the format of templates. It also natively supports interactive sample-based rule authoring. A typical usage scenario is as follows. For a new message, it is first shown to a user who can highlight the part of the message to be used for the value of a specific attribute. Then, derived from the user’s directive, Rule Builder constructs the parser rule based on conventions and heuristics. This component generates new rules in two ways: using user defined set of criteria, or using a set of keywords that are provided by the domain experts, for example keywords such as variation of “return code”, “class name”, “terminated”, “out off A”, etc. The keywords are used not only to identify messages by specific pattern of keywords, but also by lexical association. For example “Start server XYZ” may imply that there should be a “Stop Server XYZ”. Finally, a user can try out the parser rule and view its parsing results. If the results are not satisfactory, the user can further modify the parser rule either directly or by refining his previous directives iteratively.

3. Configuration Description Language

Generic Adapter uses configuration description language for description of parsing rules, filtering rules and parameters of the system. Parsing rules language (in part similar to subset of regular expression) allows to provide self-optimization during run-time processing as well as simplifies case base rule generation during off-line rule generation process. It based on generic log data format. Filtering and parsing rules allow to use reach optimization techniques.

4. Future Work

Our future work will continue along three main directions. We intend to

- introduce better optimization for regular expressions and filtering
- use text mining approach to generate message identifiers and situation types
- extend Generic Adapter with custom sensors and parsers

References

[5] Generic Adapter Logging Toolkit, accepted for publication