DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-2005-07

THE DYNAMIC DEPENDENCE MANAGER PATTERN

HELVES DOMINGUES AND MARCO A. S. NETTO

Dezembro de 2005

The Dynamic Dependence Manager Pattern

Helves Domingues, Marco A. S. Netto

Department of Computer Science
Institute of Mathematics and Statistics
University of São Paulo

{helves, netto}@ime.usp.br

Technical Report RT-MAC-2005-07

1 Objective

The Dynamic Dependence Manager pattern provides an explicit representation of run-time dependencies (i.e. dynamic dependencies) in component-based systems to allow the software dynamic extension and reconfiguration.

2 Motivation

Due to the increase of software systems complexity, software developers tend to rely on already available components to reduce the development time and improve software reliability. Therefore, a software system may depend on components and these components may rely on other components as well, generating a graph of inter-dependencies among its components.

Until recently, highly dynamic environments with mobile computers, active spaces, and ubiquitous multimedia were only present in science fiction stories or in the minds of visionary scientists. But now, they are becoming a reality and one of the most important challenges they pose is the proper management of dynamism [1]. Computer systems are becoming more dynamic in the sense that their components may be dynamically replaced, and new components may be added to or removed from the systems at execution time. In addition, they must be able to react to changes in the environment by dynamically reconfiguring themselves to keep working properly and with good performance.

Maintaining component dependencies is a complex task, in particular when the dependencies should be considered at execution time. Components are often developed by different groups with different methodologies.

An approach to manage the component dependencies is through their explicit representation. Different from static dependencies, where a representation of dependencies is utilized at system load time, the **Dynamic Dependence Manager** focuses on dynamic component dependencies of computer systems that must be reconfigured at execution time.

3 Context

Component-based systems that present an inter-dependency among components and need to deal with dynamic changes on such components. Note that this pattern does not focus on problems such as component life-cycle and discovery - these problems are discussed in the related patterns section.

4 Problem

How to manage dynamic dependencies in component-based systems?

5 Forces

Managing dynamic dependencies among components involves balancing the following forces:

- Ignoring totally the system dependencies may increase the system performance, as well as simplify the software development; but not treating dynamic dependencies properly may lead to system failure at runtime;
- In dynamic environments, we expect changes in dependencies to occur at execution time. Thus, the dependence manager must be able to deal with the dependency relationship among the components without interrupting the system completely or reducing the quality of service;
- A dependence manager could intercept all the method calls between the components and then perform management tasks based on these calls by, for example, keeping the dependencies, generating events and dealing with them. However, intercepting all method calls would lead to a large runtime overhead.

6 Solution

The management of dynamic dependencies in component-based systems is achieved with the explicit representation of the dynamic dependencies through special objects attached to each relevant component at execution time. These objects are called Dynamic Dependence Managers; they are responsible for reifying the runtime dependencies for a certain component and for implementing policies to deal with events coming from other components.

The dependencies of a component C are managed by a dynamic dependence manager C^{ddm}. Each manager C^{ddm} has a set of named *dependencies* to which other managers can be attached. These are the managers for the components on which C depends; they are called *the C dependencies*. The components that depend on C are called *dependent components*; C^{ddm} also keeps a list of references to the dependent managers. In general, every time one defines that a component C_1 depends on a component C_2, the system should perform two actions:

1. Add C_2^{ddm} to the list of dependencies of C_1^{ddm} and
2. Add C_1^{ddm} to the list of dependents of C_2^{ddm}.

Dynamic dependence managers are also responsible for distributing events across the inter-dependent components that may affect the proper system function and may require reconfiguration actions to be taken. Examples of common events are the failure or destruction of a component, internal reconfiguration of a component, or replacement of the implementation of a dependency component. The rationale is that such events may affect all the dependent components. The Dynamic Dependence Manager is the place where programmers must insert the code to deal with these configuration-related events so that system stability is maintained in the presence of dynamic changes.

Component developers can program specialized versions of the Dynamic Dependence Manager that are aware of the characteristics of specific components. These specialized managers can, therefore, implement customized policies to deal with component dependencies in application-specific ways.

7 Structure

The proposed solution for the dependence management relies on the interaction among the dependent and dependency components. Figure 1 illustrates the Dynamic Dependence Manager (DDM) object diagram, with the components and their interactions.

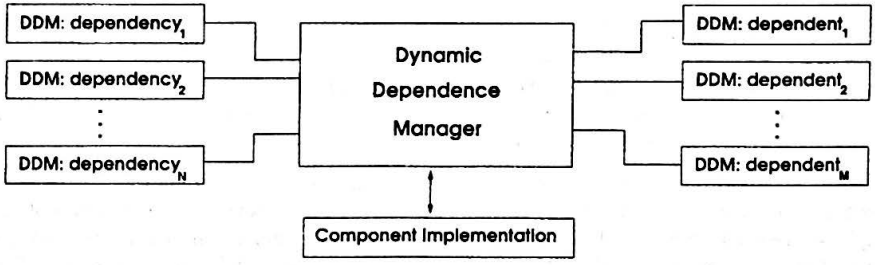

Figure 1: Dynamic Dependence Manager object diagram.

The Dynamic Dependence Manager is flexible in the sense that a dependent component may rely on several dependencies and a dependency may serve several dependents. Each Dynamic Component Manager holds a reference to the implementation of the component it manages. The UML diagram in Figure 2 shows all the class associations that represent these features.

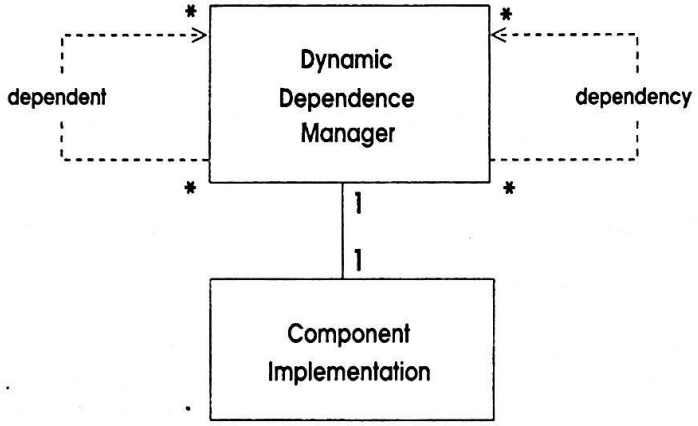

Figure 2: Dynamic Dependence Manager Class diagram.

8 Example

Consider an application responsible for controlling the products in a virtual store that sells books for thousand of people daily. One of its requirements is that it must be on-line all the time.

In this application there is a list of dynamic dependencies comprising the following components: a TCP/IP service, a local file service, and a Java-based graphical user interface. For each of these components and for the application, there must be a Dynamic Dependence Manager (DDM) to provide an explicit representation of the dynamic dependencies to allow the software extension, as well as reconfiguration.

In this scenario, suppose that the system administrator desires to upgrade the Java Virtual Machine (JVM) in this environment since a new feature to improve the performance of the graphical user interface (GUI) is available. As the GUI depends on this JVM, and the application depends on the GUI, the DDM of the GUI should notify the application DDM that the GUI will be unavailable. After that, the JVM is replaced and the administrator loads the GUI. The DDM of the user interface then notifies the application DDM that the former is available again.

In this example it is possible to observe that the application service was not interrupted and hence the users were still able to buy books. This was due to the fact that the application was notified and did not execute method calls to the graphical user interface during the JVM upgrade process.

9 Consequences

- Benefits
  - Separation of concerns: using the Dynamic Dependence Manager, application developers can focus on implementing application functionality without having to worry about managing the dependencies. Furthermore, if the developers desire, they can write the code that deals with dependencies detached from application code;
  - Flexibility to run-time changes: the dependence management allows components to be migrated, removed, added and reconfigured at run-time.
- Liabilities
  - Increase the system complexity: the developers should concern not only with the application, but also with the procedures such as the discovery of dependencies, the treatment of events, among others.

10 Implementation

The Dynamic Dependence Manager can be used both in centralized and distributed applications. For centralized applications it is used through pointers to the components, and for distributed, the DDMs could be CORBA objects or Java objects accessible via RMI.

It is interesting to use a framework as the implementation so as to reduce the user development time. When the user needs to include something specific for the application, he/she can create subclasses and implement whatever necessary.

Each relevant component has a reference to a DDM instance, where the developer includes the implementation to deal with the dependence events on a component specific way. This is essential since each DDM might need to execute different procedures depending on the event and on the component managed by such DDM.

10.1 Interfaces

Figure 3 defines an interface for objects representing the dependencies among components. A description of each method is presented below.

addDependency Adds a dependency name to the DDM.

removeDependency Removes a dependency name from the DDM.

registerDependency Adds a new DDM to the list of dependencies, associated with a dependency name.

unregisterDependency Detaches one dependency from the given dependency name.

```
public interface DynamicDependenceManager {


    public void addDependency (String dependencyName)
          throws ElementExists;

    public void removeDependency (String dependencyName)
          throws DependencyVacant, NotFound;

    public void registerDependency (DynamicDependenceManager dependency,
                  String dependencyName)
           throws ElementExists;

    public void unregisterDependency (DynamicDependenceManager dependency,
                  String dependencyName)
           throws NotFound;

    public void eventFromDependency (DynamicDependenceManager
                  dependency, ComponentEvent e);


    public void removeDependent (String dependentName)
           throws DependentVacant, NotFound;

    public void registerDependent (DynamicDependenceManager dependent)
          throws ElementExists;

    public void unregisterDependent (DynamicDependenceManager dependent,
                  String dependentName)
          throws NotFound;

    public void eventFromDependent (DynamicDependenceManager
                dependentComponent, ComponentEvent e);

};
```

Figure 3: Dynamic Dependence Manager Interface.

eventFromDependency Processes the event generated from one dependency.

registerDependent Adds a new component to the list of dependent component, linked with a dependent name.

unregisterDependent Detaches one dependent component from the given dependent name.

eventFromDependent Processes the event generated from the dependent component.

11 Known Uses

Smalltalk [2] has a dependence mechanism that is implemented in the Object class. Any object, say A, in Smalltalk can register as a dependent of another object, say B. Thus, when B changes its state, it calls a method that is implemented in the Object class. This method can use the list of all registered dependent objects of B, so it can inform to all that B has changed its state. The Smalltalk dependence mechanism has the same design as the Dynamic Dependence Manager. The difference is where the implementation is located; Smalltalk has a super class from which all others subclass (the Object class). The Dependence Manager has a component isolated from other components that has all implementation related to dependence management.

In the **Scalable Multimedia Distribution System** presented in [3], the authors describe an architecture for building scalable and extensible multimedia distribution systems. A key component of this architecture is the reflector, that acts as a relay, receiving input data packets from a list of sources and forwarding these packets to other reflectors or to programs executed by end-users. The distribution system is composed of a network of reflectors that work together to distribute the multimedia data over networks. The architecture supports fault-tolerance by using the Dynamic Dependence Manager to represent the dependencies between reflectors. When failures occur, the system uses the dependence information to locate alternate routes and keep the system working without compromising the end-user Quality of Service.

SIDAM [4] is a prototype distributed information service for disseminating road traffic information that can be queried and updated by mobile users carrying laptop computers, PDAs, or WAP-enabled cellular phones. The system should be deployable in hundreds or thousands of nodes in a metropolitan network and should be able to run continuously for months or years. Thus, the service architecture must support evolution by allowing dynamic reconfiguration and replacement of its components. Dynamic reconfiguration is supported by Dynamic Dependence Manager associated with key components of the architecture.

12 Related Patterns

The **Observer** [5] is a design pattern used to observe the state of an object in a program. The essence of this pattern is that one or more objects, called observers or listeners, are registered (or register themselves), to observe an event which may be raised by the observed

object (the subject). The subject, which may raise an event, maintains a collection of its observers.

In the **Publisher/Subscriber** pattern [6], a component takes the role of the publisher and all other components that desire to be notified about changes of the publisher information are its subscribers. For one component to be a subscriber, it must register itself through an interface offered by the publisher. In the context of the dependency manager, the subscribers are dependents of the publisher in relation to changes in it. In this sense, the dynamics of the Publisher/Subscriber and of the Dynamic Dependence Manager are similar.

The **Service Configurator** [7] pattern is also related to the dynamic configuration of systems. The main purpose of this pattern is to decouple service implementations from configuration issues when such services are initialized, finished, suspended, or resumed. As the services can be treated as modular components that can be integrated, a network of dependencies may be managed by the Dynamic Dependence Manager. Moreover, as the Dynamic Dependence Manager is only responsible for managing dependencies, the Service Configurator can assist the Dynamic Dependence Manager by loading and removing components dynamically. Thus, users can make use of both patterns to develop component-based systems.

References

[1] Fabio Kon, Jeferson Roberto Marques, Tomonori Yamane, Roy H. Campbell, and M. Dennis Mickunas. Design, implementation, and performance of an automatic configuration service for distributed component systems. *Software - Practice and Experience*, 35(7):667–703, 2005.

[2] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

[3] Fabio Kon, Roy H. Campbell, and Klara Nahrstedt. Using dynamic configuration to manage a scalable multimedia distribution system. *Computer Communications*, 24(1):105–123, 2001.

[4] M. Endler, D.M. da Silva, F. Silva e Silva, R.A. da Rocha, and M.A. de Moura. Project SIDAM: Overview and Preliminary Results. In *Anais do 2o. Workshop de Comunicação sem Fio (WCSF), Belo Horizonte*, May 2000.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1994.

[6] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented Software Architecture: A System of Patterns*, volume 1. Wiley, 1996.

[7] Prashant Jain and Douglas C. Schmidt. Service Configurator: A Pattern for Dynamic Configuration of Services. In *Proceedings of the Third USENIX Conference on Object-Oriented Technologies (COOTS)*, pages 209–220, Portland, Oregon, USA, 1997. USENIX.

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 2000 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail (mac@ime.usp.br).

Carlos Humes Junior, Paulo J. S. Silva e Benar F. Svaiter
SOME INEXACT HYBRID PROXIMAL AUGMENTED LAGRANGIAN ALGORITHMS
RT-MAC-2002-01 – Janeiro 2002, 17 pp.

Roberto Speicys Cardoso e Fabio Kon
APLICAÇÃO DE AGENTES MÓVEIS EM AMBIENTES DE COMPUTAÇÃO UBÍQUA.
RT-MAC-2002-02 – Fevereiro 2002, 26 pp.

Julio Stern and Zacks
TESTING THE INDEPENDENCE OF POISSON VARIATES UNDER THE HOLGATE BIVARIATE DISTRIBUTION: THE POWER OF A NEW EVIDENCE TEST.
RT- MAC – 2002-03 – Abril 2002, 18 pp.

E. N. Cáceres, S. W. Song and J. L. Szwarcfiter
A PARALLEL ALGORITHM FOR TRANSITIVE CLOSURE
RT-MAC – 2002-04 – Abril 2002, 11 pp.

Regina S. Burachik, Suzana Scheimberg, and Paulo J. S. Silva
A NOTE ON THE EXISTENCE OF ZEROES OF CONVEXLY REGULARIZED SUMS OF MAXIMAL MONOTONE OPERATORS
RT- MAC 2002-05 – Maio 2002, 14 pp.

C.E.R. Alves, E.N. Cáceres, F. Dehne and S. W. Song
A PARAMETERIZED PARALLEL ALGORITHM FOR EFFICIENT BIOLOGICAL SEQUENCE COMPARISON
RT-MAC-2002-06 – Agosto 2002, 11pp.

Julio Michael Stern
SIGNIFICANCE TESTS, BELIEF CALCULI, AND BURDEN OF PROOF IN LEGAL AND SCIENTIFIC DISCOURSE
RT- MAC – 2002-07 – Setembro 2002, 20pp.

Andrei Goldchleger, Fabio Kon, Alfredo Goldman vel Lejbman, Marcelo Finger and Siang Wun Song.
INTEGRADE: RUMO A UM SISTEMA DE COMPUTAÇÃO EM GRADE PARA APROVEITAMENTO DE RECURSOS OCIOSOS EM MÁQUINAS COMPARTILHADAS.
RT-MAC – 2002-08 – Outubro 2002, 27pp.

Flávio Protasio Ribeiro
OTTERLIB – A C LIBRARY FOR THEOREM PROVING
RT- MAC – 2002-09 – Dezembro 2002 , 28pp.

Cristina G. Fernandes, Edward L. Green and Arnaldo Mandel
FROM MONOMIALS TO WORDS TO GRAPHS
RT-MAC – 2003-01 – fevereiro 2003, 33pp.

Andrei Goldchleger, Márcio Rodrigo de Freitas Carneiro e Fabio Kon
GRADE: UM PADRÃO ARQUITETURAL
RT- MAC – 2003-02 – março 2003, 19pp.

C. E. R. Alves, E. N. Cáceres and S. W. Song
SEQUENTIAL AND PARALLEL ALGORITHMS FOR THE ALL-SUBSTRINGS LONGEST COMMON SUBSEQUENCE PROBLEM
RT- MAC – 2003-03 – abril 2003, 53 pp.

Said Sadique Adi and Carlos Eduardo Ferreira
A GENE PREDICTION ALGORITHM USING THE SPLICED ALIGNMENT PROBLEM
RT- MAC – 2003-04 – maio 2003, 17pp.

Eduardo Laber, Renato Carmo, and Yoshiharu Kohayakawa
QUERYING PRICED INFORMATION IN DATABASES: THE CONJUNTIVE CASE
RT-MAC – 2003-05 – julho 2003, 19pp.

E. N. Cáceres, F. Dehne, H. Mongelli, S. W. Song and J.L. Szwarcfiter
A COARSE-GRAINED PARALLEL ALGORITHM FOR SPANNING TREE AND CONNECTED COMPONENTS
RT-MAC – 2003-06 – agosto 2003, 15pp.

E. N. Cáceres, S. W. Song and J.L. Szwarcfiter
PARALLEL ALGORITMS FOR MAXIMAL CLIQUES IN CIRCLE GRAPHS AND UNRESTRICTED DEPTH SEARCH
RT-MAC – 2003-07 – agosto 2003, 24pp.

Julio Michael Stern
PARACONSISTENT SENSITIVITY ANALYSIS FOR BAYESIAN SIGNIFICANCE TESTS
RT-MAC – 2003-08 – dezembro 2003, 15pp.

Lourival Paulino da Silva e Flávio Soares Corrêa da Silva
A FORMAL MODEL FOR THE FIFTH DISCIPLINE
RT-MAC-2003-09 – dezembro 2003, 75pp.

S. Zacks and J. M. Stern
SEQUENTIAL ESTIMATION OF RATIOS, WITH APPLICATION TO BAYESIAN ANALYSIS
RT-MAC – 2003-10 - dezembro 2003, 17pp.

Alfredo Goldman, Fábio Kon, Paulo J. S. Silva and Joe Yoder
BEING EXTREME IN THE CLASSROOM: EXPERIENCES TEACHING XP
RT–MAC – 2004-01-janeiro 2004, 18pp.

Cristina Gomes Fernandes
MULTILENGTH SINGLE PAIR SHORTEST DISJOINT PATHS
RT-MAC 2004-02 – fevereiro 2004, 18pp.

Luciana Brasil Rebelo
ÁRVORE GENEALÓGICA DAS ONTOLOGIAS
RT- MAC 2004-03 – fevereiro 2004, 22pp.

Marcelo Finger
TOWARDS POLYNOMIAL APPROXIMATIONS OF FULL PROPOSITIONAL LOGIC
RT- MAC 2004-04 – abril 2004, 15pp.

Renato Carmo, Tomás Feder, Yoshiharu Kohayakawa, Eduardo Laber, Rajeev Motwani, Liadan O` Callaghan, Rina Panigrahy, Dilys Thomas
A TWO- PLAYER GAME ON GRAPH FACTORS
RT-MAC 2004-05 – Julho 2004

Paulo J. S. Silva, Carlos Humes Jr.
RESCALED PROXIMAL METHODS FOR LINEARLY CONSTRAINED CONVEX PROBLEMS
RT-MAC 2004-06-setembro 2004

Julio M. Stern
A CONSTRUCTIVIST EPISTEMOLOGY FOR SHARP STATISTICAL HYPOTHESES IN SCIENTIFIC RESEARCH
RT-MAC 2004-07- outubro 2004

Arlindo Flávio da Conceição, Fábio Kon
O USO DO MECANISMO DE PARES DE PACOTES SOBRE REDES IEEE 802.11b
RT-MAC 2004-08 – outubro 2004

Carlos H. Cardonha, Marcel K. de Carli Silva e Cristina G. Fernandes
COMPUTAÇÃO QUÂNTICA: COMPLEXIDADE E ALGORITMOS
RT- MAC 2005-01 – janeiro 2005

C.E.R. Alves, E. N. Cáceres and S. W. Song
A BSP/CGM ALGORITHM FOR FINDING ALL MAXIMAL CONTIGUOS SUBSEQUENCES OF A SEQUENCE OF NUMBERS
RT- MAC- 2005-02 – janeiro 2005

Flávio S. Corrêa da Silva
WHERE AM I? WHERE ARE YOU?
RT- MAC- 2005-03 – março 2005, 15pp.

Christian Paz-Trillo, Renata Wassermann and Fabio Kon
A PATTERN-BASED TOOL FOR LEARNING DESIGN PATTERNS
RT- MAC – 2005-04 – abril 2005, 17pp.

Wagner Borges and Julio Michael Stern
ON THE TRUTH VALUE OF COMPLEX HYPOTHESIS
RT- MAC – 2005-05 – maio 2005, 15 pp.

José de Ribamar Braga Pinheiro Jr., Alexandre César Tavares Vida and Fabio Konl
IMPLEMENTAÇÃO DE UM REPOSITÓRIO SEGURO DE APLICAÇÕES BASEADO EM GSS – PROJETO TAQUARA
RT- MAC – 2005-06 – agosto 2005, 21 pp.

Helves Domingues and Marco A. S. Netto
THE DYNAMICDEPENDENCE MANAGER PATTERN
RT – MAC 2005-07 – DEZEMBRO 2005, 12 PP.