International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT228626

An Architectural Pattern for Dynamic Component Integration

Dr. L. Kathirvelkumaran[1], Dr. R. Saravana Moorthy[2]

[1]Assistant Professor and Head, Department of Computer Science with Data Analytics, Kongunadu Arts and Science College, Coimbatore, Tamil Nadu, India

[2]Associate Professor and Head Department of Computer Science, Kongunadu Arts and Science College, Coimbatore, Tamil Nadu, India

Article Info

Publication Issue :
Volume 9, Issue 1
January-February-2023

Page Number : 101-105

Article History
Accepted: 01 Jan 2023
Published: 25 Jan 2023

ABSTRACT

The continuing increase of interest in Component-based Software Engineering (CBSE) signifies the emergence of a new development trend within the software industry. CBSE deals with the usage of reusable software components to provide complex integrated solutions at shorter development time and minimum cost. This engineering approach emphasizes the identification, selection, evaluation, procurement, integration, and evolution of reusable components for constructing complex and large-scale software solutions. Component-based development approach has a great potential for significantly reducing the cost and time to market of large-scale and complex software systems, improving system maintainability and flexibility. However the usage of components imposes some problems in the development process. Especially, the complexity of the integration process is increased due to the black box nature of the products. Integration is further influenced by problems due to architectural mismatches, component incompatibility etc. Apart from these general static integration problems, necessity for intelligent services and context aware computing paves way for the dynamic integration of components. Patterns have promising benefits in solving software design problems and their benefits have been realized in the design of component integration also. Patterns exist in different levels. Architectural patterns provide the overall structure of a system; design patterns deal with localized design choices; Idioms define patterns of usage within a particular language. Design patterns express proven techniques making it possible to reuse successful designs and provide a common vocabulary to share design descriptions. Architectural patterns cover a wider realm, specifying system-wide structural properties for an application, and impacting on subsystem architecture. This report presents the study of literature related to patterns and its application in the design of component-based systems. An Architectural pattern has been proposed in this report, for the design of a component-based system that dynamically integrates components. The dynamism is achieved using a service model.

Keywords: Component-based Software Engineering (CBSE) ,identification, selection, evaluation, procurement, integration and evolution .

I. INTRODUCTION

Component-based software development, a rapidly emerging trend in industrial software engineering, is based on the concept of building software systems by selecting, adapting and integrating a set of pre-engineered and pre-tested reusable software components.

1.1 Component Integration

A recurring problem in component based development is; there is no single off-the-shelf component that satisfies all the software requirements. In many cases, the solution is to integrate two or more components, and when combined, they satisfy the requirements, though some of the code must still be implemented in-house.

The integration approach to component usage involves the developer buying two or more separate software packages and integrating them into a larger system. Integration is influenced by problems due to architectural mismatches, component incompatibility etc. The scenario becomes more complicated when necessity for intelligent services and context aware computing arrives that paves way for the dynamic integration of components.

1.2 Software Patterns

Patterns for software development are one of the latest "hot topics" to emerge from the object-oriented community. They are a literary form of software engineering problem-solving discipline that has its roots in a design movement of the same name in contemporary architecture, literate programming, and the documentation of best practices and lessons learned in all vocations.

The authors of Patterns of Software Architecture[] define three types of patterns: architectural, design and idiom.

An architectural pattern expresses a fundamental structural organization or schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

A *design pattern* provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes commonly recurring structure of communicating components that solves a general design problem within a particular context.

An *idiom* is a low-level pattern specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language.

1.3 Architectural patterns

Architectural patterns have been proposed in many domains as means of capturing recurring design problems that arise in specific design situations. They document existing, well-proven design experience, allowing reuse of knowledge gained by experienced practitioners [1]. For example, a software architecture pattern describes a particular recurring design problem and presents a generic scheme for its solutions.

The solution scheme contains components, their responsibilities and relationships. Patterns for software architectures also exhibit other desirable properties: (i) patterns provide a common vocabulary and understanding for design principles; (ii) they are a means for documenting software architectures; (iii) they support the construction of software with defined properties; (iv) they support building complex and heterogeneous software architectures; and (v) they help managing software complexity.

1.4 Patterns and component based design

The prospect of building software applications out of pre-fabricated parts represents a fundamental departure from the development-centric view of current software engineering practices. This is an assembly-based and composition-centric view of software engineering (Component-based Software Engineering).

Component-based software development encounters several major challenges, one of which is to integrate components in software systems because it is not easy to modify components and most components are black boxes. Hence, selecting appropriate components and identifying the interactions among these components become vital to the success of component-based software development. The integration mechanism has to perform compatibility check of a black-box component with a target architecture based on functional and non-functional requirements. Before components are integrated, the way in which components are expected to interact among themselves has to be known. Some interactions result in better software quality and others do not. Hence, it is important to identify component interactions in a way to achieve good software quality and high development productivity. Design patterns incorporate proven design experiences and reusing them will prevent designers from discovering solutions to each solution problem from the scratch. Therefore an approach has been proposed [30] that use design patterns for automatic generation of the component wrappers for component integration. The goal is to facilitate CBSD by partially automating component-based software design and implementation.

The component-based software development process that adopts design patterns can be summarized in Figure 1.

1.5 Dynamic Integration of Components

Traditionally, in CBSE technologies, several components are packaged together to create software systems. Emerging concerns include multiple suppliers of components that provide the same functionality, coping with multiple versions, and configuration of components. Currently, CBSE addresses issues and technologies related to COTS components, inbuilt components, and application frameworks.

Flexible information systems use COTS components because they cost-effectively supply required component functionality. However, as there is a movement toward a world in which programmable devices greatly outnumber people, information systems will increasingly need to address this ubiquitous-computing context, or ambient intelligent environment. Such dynamic environments require coping with anticipated change, such as the release of new COTS versions. However, they also require coping with emergent behavior, which arises from interactions between a system’s components (including its environment) and thus can’t always be anticipated.

Thus the integration of COTS components into such a system requires a dynamic architecture that changes as the system evolves. This runtime architecture can integrate COTS components into flexible information system. Such a system exists in a dynamic environment and must evolve to incorporate new capabilities from that environment. Thus a set of mechanisms for integrating COTS components and incorporating control systems principles are needed to guide the system’s ongoing evolution which extends its use of COTS components’ capabilities, replace one COTS component with another, or refine its own architecture based on the COTS components available in its environment.

To summarize, dynamic component integration facilitates the creation of so-called Intelligent Services from COTS components, legacy components, and application frameworks. These services can be integrated at run time based on available resources and thereby allow for a federation of services that can evolve over time.

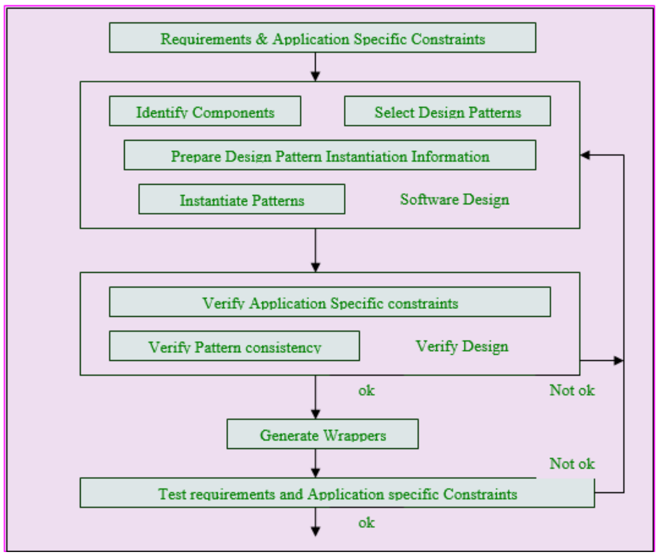
Requirements & Application Specific Constraints
Identify Components
Select Design Patterns
Prepare Design Pattern Instantiation Information
Instantiate Patterns
Software Design
Verify Application Specific constraints
Verify Pattern consistency
Verify Design
ok
Not ok
Generate Wrappers
Not ok
Test requirements and Application specific Constraints
ok

Fig. 1. Pattern based development of CBSD

II. AN ARCHITECTURAL PATTERN FOR DYNAMIC COMPONENT INTEGRATION

Context

Software systems mainly information systems use components because they cost-effectively supply the required functionality. The whole system is designed by integrating a number of heterogeneous (desktop, legacy, network) components so as to offer the expected functionality. However, as there is a movement toward a world in which programmable devices greatly outnumber people, information systems also need to be flexible in order to address the rise of ubiquitous-computing context, or ambient intelligent environment. Such dynamic environments require coping with anticipated change, according to users requirements. However, they also require coping with emergent behavior, which arises from interactions between a system's components and thus can't always be anticipated. This requires a dynamic environment in which components are to be integrated at run time.

Problem

Emerging technologies of component integration provide a component model where a predefined infrastructure acts as "plumbing" that facilitates communication between components. Hence a novel approach is needed that supports the dynamic integration of components.

Forces

Dynamic integration requires the dynamic discovery of the interfaces, handling of mismatches between the components while integration and the automatic generation of glue code so as to integrate the components at run time.

Solution

Given these software engineering considerations, our goal is to provide a more flexible framework for integrating software components. We are making use of the Service Model in order to facilitate the dynamic integration of components. From an architectural point of view the proposed system is a networked application, consisting of interacting service components. Components can be integrated into the system at any time and integrated components can also be disconnected from the system after the usage.

The service-based paradigm used for dynamic component integration looks at software reuse from a different perspective in which components are viewed as services available on a network The proposed framework supports the construction of applications by dynamically integrating these services at run-time, based on available resources that allows for a federation of services that can evolve over time.

The process of dynamic component integration involves wrapping of components of services with appropriate adapters and storing them in a repository.

Generation and execution of clients performs the integration of the specified services at run time.
Once the services are generated and stored in a repository, clients can make use of these services by identifying a general class of service that is required and the architectural style is necessary to properly interact with a client.
Once services become available and join the network, the client is notified. The client integrates with the services by performing the GUI component integration as well as the service adapter integration and utilizes the service.

Solution Discussion
As the components of the service model implements only a portion of the required functionality, it can be smaller and, therefore, easier to develop, test, and maintain. The memory requirements, and consequently the execution speed, of software applications can be reduced. This is justifiable because an application need contain only the core functionality, while other, more exotic features, can be loaded and run only when needed. As the components that provide services are loaded on demand and removed afterwards, the process of updating the functionality of an application can be simplified. New versions of those components can be substituted as soon as they are developed; whenever the user needs specific service or services, he will access the most up-to-date version.

Finally, more flexible pricing schemes can be devised and implemented, in which the users pay only for the services (i.e., functionality) they actually use, rather than for the mammoth-sized applications with the functionality they don't need and never use.

Other potential benefits of using a service-based approach for developing software is that at any given time, a wide variety of alternatives may be available that meet the needs of a given client (availability). As a result, any or all of the services may be integrated with a client at runtime.

Resulting Context
You now have a flexible framework which facilitates dynamic component integration thereby allowing the evolution of a federation of services.

Known Uses
The know uses of dynamic component integration using a services model include systems for context aware computing, ambient intelligence and exploitation of services in ad-hoc networks.

Diagram

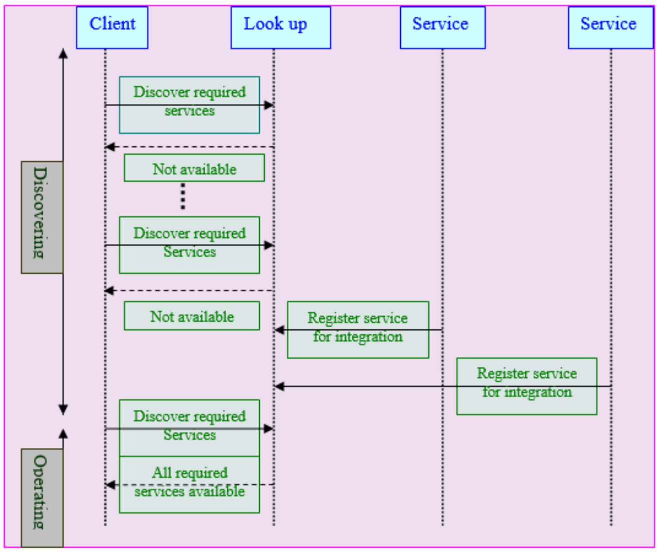
Client
Look up
Service
Service
Discovering
Operating
Discover required services
Not available
Discover required Services
Not available
Register service for integration
Register service for integration
Discover required Services
All required services available

Fig.2 Dynamic discovery and integration of components

III. CONCLUSION AND FUTURE WORK

The discussed work focuses on two emerging fields of software development viz. component based software development and software patterns. The literature related to patterns and its application to components has been analyzed. An architectural pattern has been proposed for the problem of dynamic component integration. This work has been extended by formulating a pattern language for the same problem discovering solutions to the recurring problems that arise in dynamically integrating components using the services model. The proposed patterns have been used in the construction of a component based personal information system.

IV. REFERENCES

[1]. Ali Arsanjani," A Pattern-based Approach for Building Reusable, Technology-neutral Component Integration Architectures", IBM, National E-business Application Development, Center of Competency.

[2]. Ali Arsanjani, "CBDi: A Pattern Language for Component-based Development and Integration", IBM, National E-business Application Development, Center of Competency, 2001.

[3]. Andy Crabtree, Tom Rodden, "Patterns: Problem and Solutions?", The University of Nottingham.

[4]. Anind K. Dey, Gregory D. Abowd, Andrew Wood,"CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services", ACM Press, 2015.

[5]. Gerald C. Gannod, Sudhakiran V. Mudiam, Timothy E. Lindquist, "Automated Support for Service-based Software Development and Integration", The Journal of Systems and Software, 2004 Haines Capt Gary, Carney David, Foreman John, " Component-Based Software Development/ COTS Integration", Software Technology Review, 2017

[6]. Philip Eskelin, "Component Interaction Patterns", PLoP 1999.

[7]. Stephen S. Yau, Ning Dong, "Integration in Component-Based Software Development Using Design Patterns", IEEE, 2020.

[8]. Uwe Zdun, "Some Patterns of Component and Language Integration".

Cite this article as :

Dr. L. Kathirvelkumaran, Dr. R. Saravana Moorthy, "An Architectural Pattern for Dynamic Component Integration", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 9, Issue 1, pp.101-106, January-February-2023. Available at doi : https://doi.org/10.32628/CSEIT228626

Journal URL : https://ijsrcseit.com/CSEIT228626