Software development process

EddieQi*

ZheJiangNormalUniversity

Software Project Management

Abstract

A flexible component-based software system is developed by generating a UML model in a specification phase . Components are generated in an implementation phase . The UML model is transformed to a meta model. A runtime architecture service (RAS, dynamically assembles a system in runtime according to the meta model. A modelling tool modifies the meta model via an API and a meta modeller . This allows dynamic re-configuration of the system.
Keywords: waterfall, prototype, incremental, RUPIntroduction

1 Introduction

A software system is generally regarded as “flexible” if it can undergo modification with relative ease in order to accommodate new requirements. This change may be effected via a restructuring of the interactions of its functional elements and/or the replacement of existing elements with alternates or via the addition or removal of elements.

Flexibility is of benefit in a system which is required to operate in an environment which undergoes change. Not all systems are required to be flexible. However, in general, flexibility and the reduction of the cost of providing flexibility are often considered to be of benefit.

The principal difficulty in delivering flexible software has long been known: strong couplings or bindings between code modules act to limit their viable independent usage, and lock the modules together in ways that are difficult to manipulate. Object oriented technologies are based on strong couplings between objects based on the references. That these references are created and destroyed empirically, within functional code, in an intermingled fashion, further hardens the linkage. This is exacerbated by the fact that it is a compile-time linkage. This leads to high costs when restructuring interactions between tightly coupled modules. Reuse and perhaps more importantly, changing a system's behaviour is not possible without inspection and modification of code, an expensive process.

PCT Patent Specification No. WO00/14629 describes a method for generating a system, in which a meta model is initially developed. This is transformed into an application model, which is in turn transformed to code. Code for client/server interfaces is also generated. The structure of the meta model reflects that of the end-product system and it is used as a starting point, providing structure for the overall production process.

Attempts have been made to improve flexibility in software. An example is a message-based technologies in which systems are constructed as anonymous peers which exchange messages via an intermediary. Another approach is to use a naming service which allows services to be named abstractly. However, these approaches provide limited flexibility because the components have a flat peer-to-peer structure which it is difficult to implement realistic system architectures.

Another approach has been the development of Service Oriented Architecture (SOA). SOA models provide a component-oriented paradigm which views enterprise level software components as providers and consumers of business services. This is an example of component technology. However, the extent of flexibility is limited because this technology does not allow modification of interactions between components, without re-engineering at the code level.

1 Methods/Techniques

OBJECTIVES OF THE INVENTION

The invention is described towards providing a development process which produces a software system having improved flexibility.

SUMMARY OF THE INVENTION

According to the invention, there is provided a process for development of a software system, the process comprising the steps of defining a meta model and using the metal model to generate executable code, characterised in that,

-
  - the process comprises a step of storing components each having executable code;
  - the meta model has a hierarchical structure defining how a system is to be assembled by instantiating components in runtime; and
  - the system is completed by dynamically instantiating the components in runtime according to the meta model.

In one embodiment, the meta model is generated from a design model arising from a specification phase, the design model specifying the system in terms of components and their interactions.

In another embodiment, the design model is represented as a markup language file.

In a further embodiment, the meta model is saved to storage by transforming it back to an associated design model in said markup language.

In one embodiment, the design model models the system declaratively as a graph of instances of component types and allowed interactions between them.

In another embodiment, the design model models a component type as an abstraction that specifies interfaces supported by a component and interfaces that a component requires.

In a further embodiment, a component type is specified as a specialisation of another component type.

In one embodiment, a component type specifies how a plurality of component types may be aggregated to form a composite component type, and in which the composite component type is modelled as a template.

In another embodiment, channels between component instances are templated such that they can be bound to specific interface types on subsequent reuse of the templated component type.

In a further embodiment, the meta model is generated by parsing the design model to generate a hierarchical graph of connected objects.

In one embodiment, the parsing generates a token stream arising from lexical analysis, the token stream representing markup language elements and attributes.

In another embodiment, the token stream is parsed in a plurality of passes comprising:

- a syntactic parsing pass to create meta model objects;
- a linking parsing pass to create links between objects based on defined bidirectional associations;
- a pass to refine the objects according to stereotypes; and
- a pass to perform semantic model validation for correct behaviour.

In one embodiment, the stereotypes include:

- a component type stereotype;
- a realisation stereotype to extend abstraction; and
- an interface requirement stereotype.

In one embodiment, the meta model comprises:

- a type model comprising a specification of the types of components; and
- an instance model which instantiates the type model set into runtime components.

In one embodiment, the type model specifies component types in terms of interfaces and in terms of services implemented by components, and in which the interfaces are those supported by a component and those of services that component requires.

In another embodiment, the invention comprises the further step of maintaining the meta model and dynamically modifying the system by modifying the meta model, and re-instantiating the components according to the modified meta model.

In one embodiment, the meta model is modified by a modelling tool and an API, the tool having a model view pattern on the meta model.

In another embodiment, the dynamic modification of the meta model comprises dynamically changing the number and identity of component instances, component types, and their interactions.

In one embodiment, the meta model is dynamically modified by generating a fresh or modified design model, transforming the design model to a meta model, and re-assembling the system according to the new meta model.

In another embodiment, the meta model automatically performs self type-checking before assembly of the system.

In one embodiment, the system assembly is performed by a technology-independent activation manager and at least one technology-specific activator, in which the activation manager directs at least one activator to:

- instantiate components,
- bind components, and
- unbind components.

In one embodiment, the activation manager is an object which presents public methods for instantiating components, for binding components, and for unbinding components, and in which the activation manager instantiates technology-specific components by:

- retrieving a technology identifier;
- searching a list of currently instantiated technology-specific activators and identifying one having the same technology identifier; and
- said activator instantiating a component.

According to another aspect, the invention provides a process for development of a software system, the process comprising the steps of defining a meta model and using the metal model to generate executable code, characterised in that,

- the process comprises a step of storing components each having executable code;
- the meta model has a hierarchical structure defining how a system is to be assembled by instantiating components in runtime;
- the system is assembled by dynamically instantiating the components in runtime according to the meta model;
- the meta model is generated from a design model arising from a specification phase, the design model specifying the system in terms of components and their interactions;
- the meta model is maintained and the system is dynamically modified by modifying the meta model and re-instantiating the components according to the modified meta model, wherein the meta model is modified by a modelling tool and an API, the tool having a model view pattern on the meta model.

According to a still further aspect, the invention provides a software system comprising components having executable code, characterised in that the system further comprises:

- an adapter comprising means for generating a meta model from a design model, the meta model defining how the components are to be instantiated in runtime;
- a meta modeller comprising means for maintaining a meta model during runtime;
- an activation sub-system comprising means for dynamically instantiating the components in runtime according to the meta model.

In one embodiment, the system further comprises a modelling tool comprising means for interacting with the meta modeller to modify the meta model, and the activation sub-system comprises means for re-instantiating the components for dynamic modification of the system.

In another embodiment, the meta modeller comprises means for maintaining the meta model with:

- a type model comprising a specification of the types of components; and
- an instance model which instantiates the type model set into runtime components.

In one embodiment, the activation sub-system comprises a technology-independent activation manager and at least one technology-specific activator, in which the activation manager comprises mean for directing at least one activator to:

- instantiate components,
- bind components, and
- unbind components.

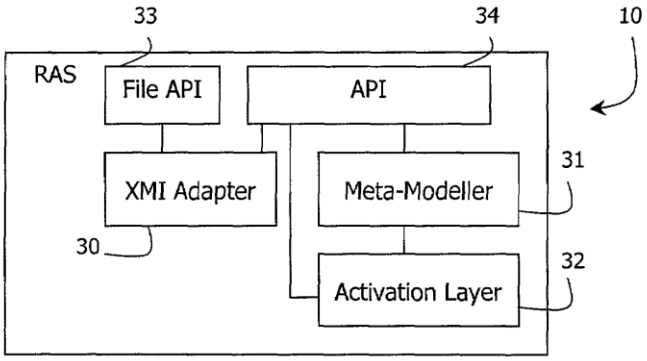
DETAILED DESCRIPTION OF THE INVENTION Brief Description of the Drawings

The invention will be more clearly understood from the following description of some embodiments thereof, given by way of example only with reference to the accompanying drawings in which

FIG. 1 is a high level diagram illustrating a software development process of the invention;

FIGS. 2 and 3 are diagrams showing execution of the system;

FIGS. 4, **5**, and **6**, are diagrams illustrating components and how they are modelled;



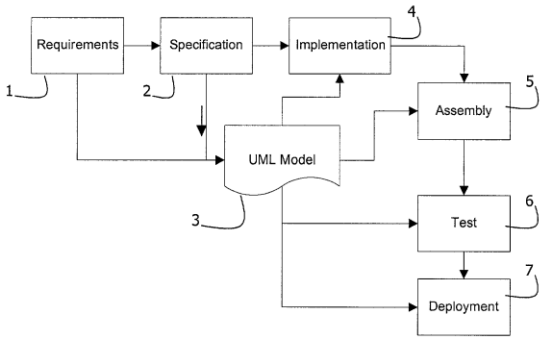


Fig. 1

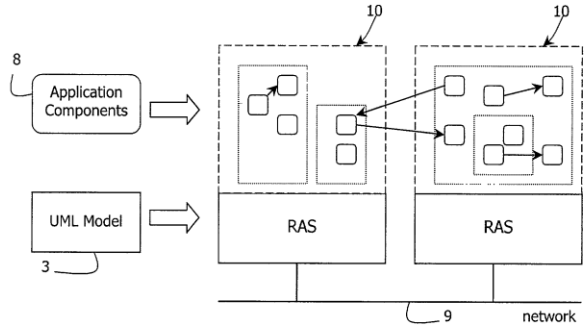


Fig. 2

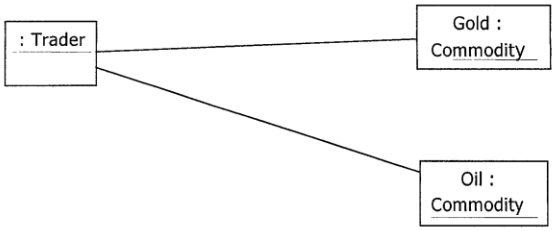


Fig. 5 (a)

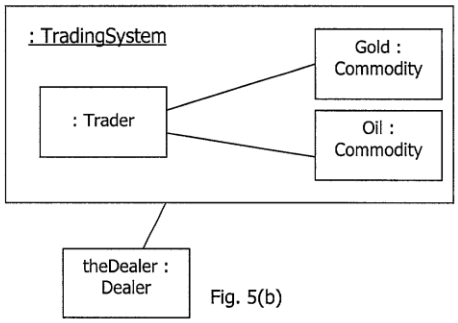
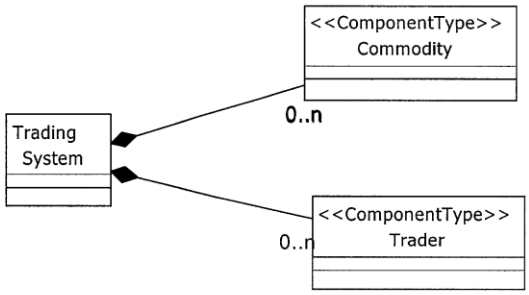


Fig. 5(b)



Component type modeled as complex component type

Fig. 6

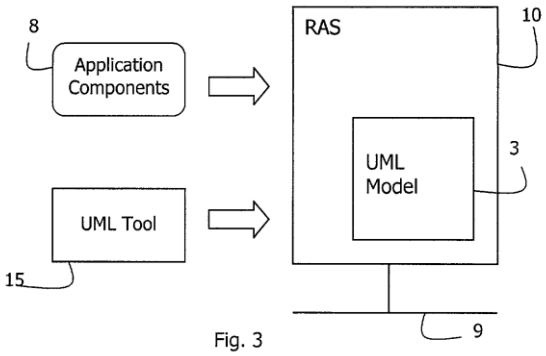
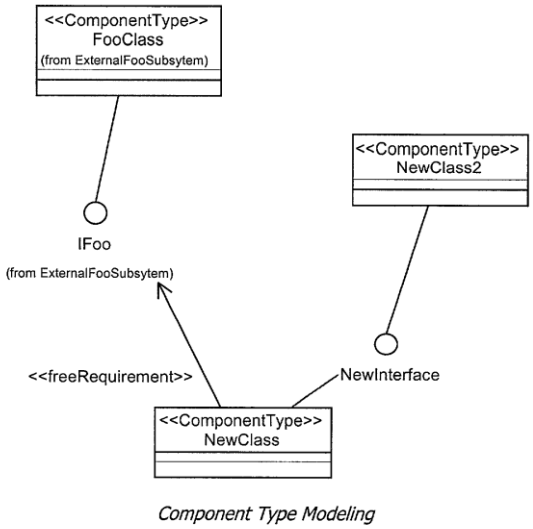


Fig. 3



Component Type Modeling

Fig. 4

References

Weikipaide