Presenting Reusable Service Models in Model-Driven Service Engineering

Selo Sulistyo

***Abstract*—Current developments of software systems show the shift towards increasingly service-based, where a software system is composed of software units called services. For software development, model-driven approaches are getting more attention since it promotes a rapid development process which increasing software productivity. To be able to apply model-driven development approaches for the development of service-based applications, presenting services and their interactions using (often) graphical presentation is required.**

This paper proposes to adapt the CORBA component model (CCM) to model a service and to present the service as a building block. In addition to use ports in CCM, we also propose to use activity diagrams to describe internal behaviors of a service and to describe interactions between building blocks. The main contribution of this work is an automated method to present services described in textual into graphical presentation for supporting model-driven development.

***Index Terms*—Model-driven development, CORBA component model.**

I. Introduction

Current developments of software systems show the shift towards increasingly service-based, where a software system is composed out of compound, heterogeneous, autonomous software components called services. With this, development focus of software systems shifts from activities concerning the in-house custom design and implementation of the service components, to activities concerning the identification, selection, and composition of services offered by third parties. In this paper we focus on the services composition.

A service can be defined in several ways. Services are also used widely from device levels to business level services. With regard to business processes, a service can be defined as a self-contained business process or service with predetermined and well-defined functionality that may be exposed through a well defined business or technology interfaces. But, from the view of software developers, a service is considered as a piece of an executable software unit that can be invoked via its interfaces.

As a unit of functionality, services can be requested, performs one or more operations, and returns a set of results to the requester. With this, the development of software systems supporting for business processes is done by composing pre-made services (provided by third parties) that can give support to real business environment which is usually changing with increasing speed. In general we call this kind of software systems as a service-based system.

Model-driven development (MDD) deals with the provision of models, transformations between them and code generators to address software development. The idea is to develop systems starting from models, at different levels of abstraction, that are capable of expressing domain specific concepts in a way that is at the same time precise, intuitive, and machine-processable, allowing automated manipulation and transformation. One example of the model-based software development approaches is OMG's Model Driven Architecture, MDA [1] that uses the Unified Modeling Language (UML) [2] as the modeling language.

The convergence of service-based systems with model-driven development approaches can holds out the promise of rapid and accurate development of software systems. With model-driven approaches, the composition of pre-made services can be done at models levels. Then, different code for different target platforms implementing the software systems can be generated automatically from the models.

Being model-driven there is a need to present services and their interactions in an abstract manner and often using graphical presentation. For this, several models to present services have been proposed. Unfortunately, there is a lack of agreement on a standard way for presenting services in model-driven environment. We can of course use UML profiles to present services in a model-driven environment. But since UML is a general modeling language, we consider that it is difficult to express accurately (semantic), for example expressing interactions between services.

This paper presents our approach of presenting for presenting services as a generic reusable service component model. We organized the rest of the paper as follows: Section 2 gives a background for the paper where we present the basic idea of the use of MDD for service engineering. In Section 3, we present our survey on approaches for presenting software component models. Next in Section 4, we propose to adapt the CCM [3] to present service component model, where we also present an example. Section 5 is devoted for related work. Finally, we draw our conclusions in Section 6.

II. Model-Driven Approaches for the Development of Service-Based Systems

Two approaches of software development exist: model-oriented using modelling languages and implementation-oriented using programming languages. In

Manuscript received May 12, 2012; revised June 24, 2012.

Selo Sulistyo is with the Department of Information Technology and Electrical Engineering, Gadjah Mada University, ll. Grafika no.2, Indonesia (e-mail: selo@ugm.ac.id).

DOI: 10.7763/IJIEE.2012.V2.189

this paper we focus on model-oriented approaches, in particular model-driven development (MDD).

A. Model-Driven Development

New ways of complexity handling take higher levels of abstraction and describe systems using models. In particular, OMG puts forward their idea of MDA, which focuses on software development by means of high-level models. Using MDA implies creation of UML models of the following kinds: the computational independent models (CIM) at business model level, the platform independent model (PIM) at information models level, the platform-specific model (PSM) at software model levels, and the code which will automatically be generated from the PSM. In this case, the models (PIM and PSM) present the structural and behavioral parts of a software system.

Three aspects for the success of model-driven development approaches are models, modeling languages and model transformation (e.g. code generators). A model is an abstract presentation of a system which describes the structure and the behavior of the system. The structure specifies what the instances of the model are; it identifies the meaningful components of the model construct and relates them to each other. In contrast, the behavioral model emphasizes the dynamic behavior of a system, which in MDA can essentially be expressed in three different types of behavioral models; UML interaction diagrams, activity diagrams, and state machine diagrams.

Obviously, expressing a software system by models requires a modeling language as a tool. Since models might have different abstraction levels, it might be a need to have different languages for each abstraction level. For example, BPMN [4] is considered as a modeling language for the highest abstraction level (business models), which in MDA would be CIM. SOAML [5] is considered as a language for modeling PIM and PSM levels. UML is an example of a general modeling language. By UML, models of a software system can be represented in different views such as UML class diagrams, activity diagrams, state machine diagrams, and collaboration diagrams at different abstraction levels.

However, in software system development, the most important is models execution to have running systems. With regard to the models execution there exist two opinions. The first opinion believes that a model must be executable. Therefore models must contain complete information for the execution. With this opinion, code is just another presentation of a model. In contrast, the second opinion believes that models do not necessary to be an executable. It is up to the developers to implement the models into an executable code. They do believe that the implementation is another case.

In line with the two opinions above, two alternatives of models execution exist. We can either develop a graphical interpreter that reads and executes the models directly or develop a code generator to execute indirectly. With a code generator, graphical models are transformed automatically into textual presentation (i.e source code). The second alternative is preferred by taking the benefit on programming languages.

B. Model-driven Approaches for the Development of Service-based Systems

A service-based system is containing one or more services where the services itself might a basic service or composite. Interactions of services determine the behavior of a service-based system. With regard to code for implementing a service-based system, it would be like a "glue code" that manages the interactions of services. For the development of this "glue code", we propose a bottom-up development in three steps; presentations, modeling and code generation, as shown in Fig. 1, below.

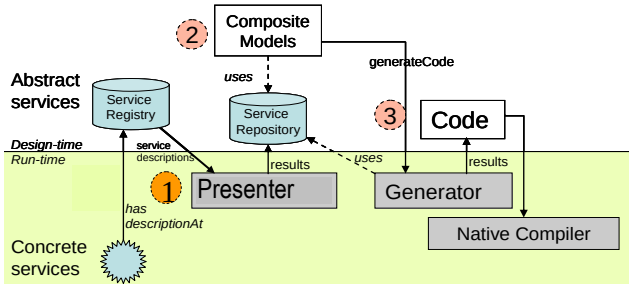

Fig. 1. The proposed steps of the software service composition

1) Presentation

The first step is to present existing pre-made services graphically that conforms to a specific modeling language. In addition to a graphical presentation, this first step should also generate an implementation class that acts as a proxy class between new software and the concrete services. This kind of re-engineering process is possible since the services are often described using open standard, for example UPnP, DPWS, WSDL, etc. Moreover, several frameworks/API for these technologies are also available, such as UPnP Cyberlink for Java [6] for UPnP, Ws4d [7] for DPWS and Axis (WSDL2Java) [8] for Web Services.

2) Modeling

After we presented all existing pre-made services and stored them in a service repository, the modeling of new software can be started. The new service-based system would be interactions of abstract services. Depending on the modeling language, different symbols and notations can be used to present these interactions. But obviously, the interactions from one abstract service to another would mean service invocations.

3) Code Generation

The last step is used to generate code that conforms to a specific programming language. Native compiler can be used to compile the generated code. Based on the modeling language chosen in the first step, code should be generated automatically.

III. Approaches for Presenting Software Component Models and Their Interactions

With regard to the managing software systems complexity, already in 1968, [9] has proposed the use of reusable software components instead of hand-coding from scratch to build software systems. Based on this proposed idea, several approaches for software component models were introduced (i.e. modules, objects, components, and the last component

models called services). In this section, we present and discuss approaches for presenting the modules, objects, components with regard to MDD. We focus on how they (and their interactions) are presented. We will also discuss how these software component models are used with regard to the bottom-up and top-down development approaches.

A. Module Models

By modules, the development of a software system is split into small and independent subsystems that are developed and tested separately, that latter will be composed. A subsystem is called a module which is obviously a software unit.

With regard to the composition language, there are no particular languages that are supposed to be languages for composition of for the modular although the concept has been used by most of general-purpose languages (i.e. C++, Java, Pascal, etc). The composition technique depends on the language but in general we use often export/import statements. A new module can import several already created modules and use them as part of the code.

For the presentation, there is no standard for presenting modules since modules are often made hand-coding (textual presentation). For this reason, the composition of modules is often done hand-coding (textually). A new module-based system is often defined in terms of implementation.

With regard to the development approaches, although top-down and bottom-up approaches can be applied, in practice only the bottom-up approaches are used. New software systems are built or composed from the implemented modules.

B. Object Models

It was believed that modular component models are inflexibility. For this reason, object-oriented systems were introduced, stating with the Simula programming language [10]. Component models in object-oriented systems are either classes at design-time or objects at runtime. An object is an instance of a class.

For the presentation, a class is often used to present objects. A class can be defined as a collection of objects that have similar properties. With UML modeling language for example, a class and an object are presented as shown in Fig. 2 As shown, a class has a name, attributes, and operations. An object is an instance of a class. An object has a delegation function, inheritance, and aggregation.

Connecting classes at design-time using associations can be considered as composition techniques. For example we can use import or extends statements to define a connection between objects. Since objects can be presented graphically using a UML class, compositions of objects can be defined graphically by defining a class diagram.

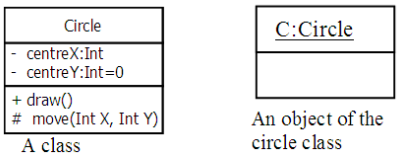

Fig. 2. An example of represnting a class and object using UML

With regard to development approaches, both top-down and bottom-up approaches can be applied. But often top-down approaches are used, where the development of software systems is started from models (e.g. UML class diagrams) that later will be implemented.

C. Component Models

A component can be defined in different way, but we consider it as other software units. The basic idea of this type component model is to completely isolate the implementation and to provide ready-to-use components off-the-shelf (COTS). These components are a packaged software component that are easy to use, easy to learn and easy to distribute. Among well known component models are OMG's CORBA, Sun's JavaBeans and Enterprise JavaBeans, Microsoft's DCOM and .NET. Fig. 3 shows an example of a presentation of software component model for CORBA Component Model (CCM).

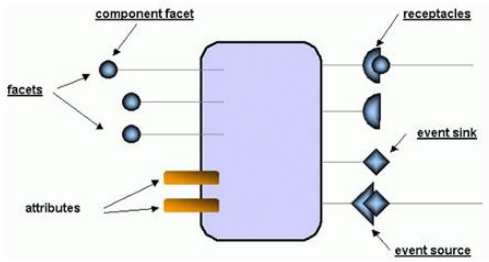

Fig. 3. An example presentation of CORBA component model

As shown in Fig. 3, the CCM have four different ports: ***1)*** Facets, interfaces that are provided for client. *2)* Attributes, that are used for configuration of installed component. *3)* Receptacles, to connect to other components, and *4)* Event sockets, that are used to handle the events.

Unlike object models where we can present using UML, in our opinion there is no specific tool or language that supports graphically for the composition of component models. There is component diagram in UML or UML profiles but they are not expressive enough. There exist also CORBA framework but most of them are textual based.

As with object component models, both top-down and bottom-up approaches can be applied for this type of component model.

IV. Service Component Models

By the concept of service-orientation system, the development of complex software systems is done often in two phases and involves two roles. First, service developers define, implement, and describe service components and second, software integrators compose the pre-made services as new service-based applications. Since it involves two roles, a proper service description is required.

A. Service Descriptions

Services can be described in many ways. Fig. 4 shows a conceptual model of services. As it can be seen in the figure, a service can be abstract or concrete, it might be a simple or composite services. It has behavior, capability, interfaces, and other properties. All these properties are described in the service description.

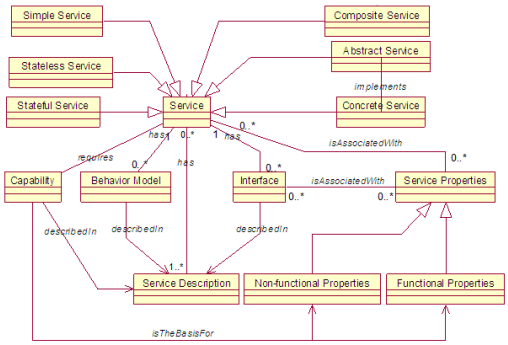

Fig. 4. A conceptual model of services

A model of a service-based system specifies the composition of services. For the composition, there exist several composition techniques such as service orchestration [11], choreography, and wiring [12]. These techniques specify interactions between services. However, there is a lack of accepted standard for presenting services and their interaction.

It is important to emphasize that, in order to be able to apply model-driven approaches for the development of service-based systems it is a need to present concrete services and their interactions using graphical presentation conforming to a specific modeling language. For example, if we want to use UML as a modeling language, it is required to present services in UML classes where we can then define interactions between these classes (creating a class diagram) to model a new application.

B. Presenting a Service with a Building Block

We propose to use a building block to present software component called a service. The idea is using the CORBA component model (see Fig. 5.) as a conceptual model of a service then presents it graphically as a building block. With the building blocks, the definition of a new software system can be done by defining a collaboration of these building blocks.

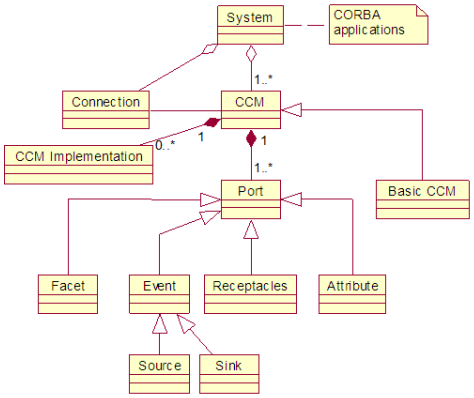

Fig. 5. A conceptual model for CCM-based systems

Referring to Fig. 5, a building block is use to present a service. A building block has ports and an internal behavior. The ports are use to access the access the internal behavior and to communicate with other building block. With this idea, a service can be considered as an encapsulation of activities that can be accessed via its interfaces.

A building block has two kinds of ports that are input ports (CallAction, EventIn, SetAttribute) and output ports (CallActionResult, EventOut). CallAction and CallActionResult ports are a pair port. When a CallAction has a result, the result will be delivered via CallActionResult port. A building block has also a description of its internal behavior (an activity diagram) that can be accessed via the ports. In addition to building blocks, the transformation also generates an implementation class.

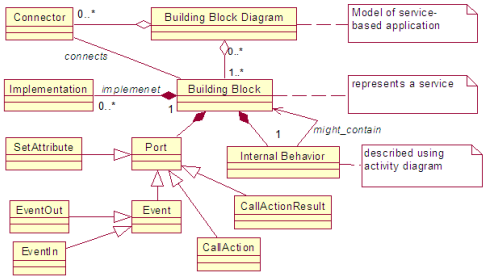

Fig. 6. An adapted CCM for presenting a service

The adaptation of the building block from the CCM follows rules as presented in Table I. As shown in the table, we use different names to call the ports. The reason is that since they are different entities then we need a different way to present them.

TABLE I: ADAPTATION TABLE FROM CCM TO BUILDING BLOCKS

CCM	Building Block
CCM identity	Service name
Facets	CallAction
Receptacles	CallActionsResult
Event source	EventOut
Event sink	EventIn
Attributes	SetAttribute

C. A Case Study: Developing a Service-Based Application

To illustrate the concept of model-driven service development we compose two different services technologies (UPnP services and Web services). For the purpose of presentation we adapted Arctis[13] to present services as reusable building blocks.

Services in a UPnP are embedded in the UPnP device. They are described in the UPnP device description. Fig. 7 below is a conceptual model for the UPnP devices. As shown in the figure, a UPnP device description has a two kind of descriptions; device and service descriptions. A UPnP can have several services that are defined in the Actions.

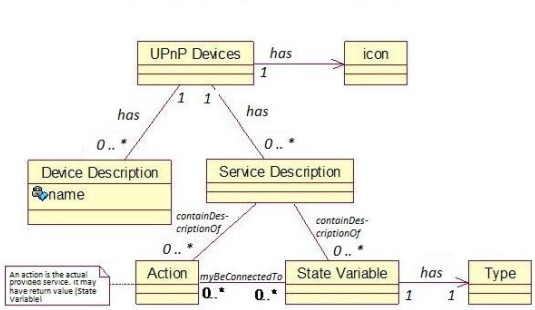

Fig. 7. A conceptual model of UPnP services

To be able to present a UPnP service as a building block we need to transform the UPnP service descriptions into a building blocks and generate proxy classes. To automate the transformation we need a transformation rule, which is

shown in Table II. For example, it will transform any actions in the UPnP service description into parameters (activity nodes) in the building block.

TABLE II: TRANSFORMATION RULES UPNP SERVICE S IN A DEVICE INTO A BUILDING BLOCK

UPnP	Building Block
Device Name	Service name
Action	CallAction
Action argument	CallAction argument
State variable	Event out
Type of state variable	Type of event out

A building block is presenting one unique UPnP device that might have several services. To present the name of the device, we use a device URI that is unique for each device. Fig. 8 shows an example of a building block as an output of the service transformer for the UPnP light service. Form more information about the service description of the UPnP light, the reader should refer to [14]. Different color for different ports (see also Fig. 8). Accordingly, we transform Web services into building block.

Fig. 8. A building block model for UPnP light service

To develop a service-based application we compose building blocks by collaboration. Fig. 9 shows an example of building blocks collaboration. In this example, we assume that there exist different services running independently. Among of them are a UPnP WeatherModule service, a UPnP light service, services from a UPnP Media Renderer, services from a UPnP Media Server and a sendSMS Web service. We want to develop a new application which is a type of event-based system. The new application enables sending notifications (SMS) when the room temperature from the WeatherModule is greater than 50 degree. In addition, the application will also play music/songs on a media renderer when the light (a lamp) is turned on and the solar radiation is below than 10.

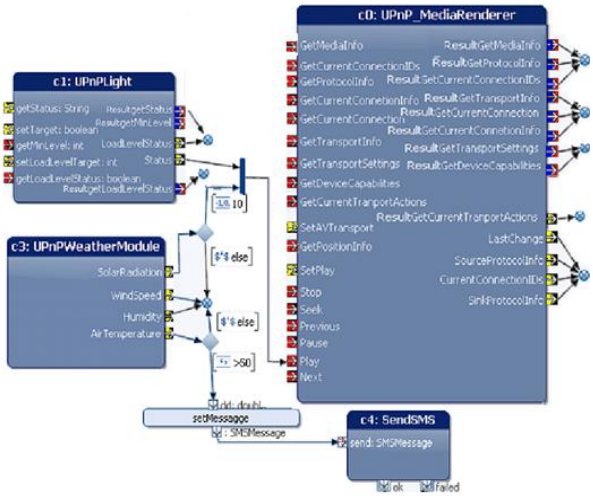

Fig. 9. A model of the new service-based sytem

As shown in the figure, service interactions (service logic) of the new service-based system are defined by defining collaboration activities of building blocks. The building blocks represent the actual services and its implementation class acts as a proxy for service invocations. From this building block diagram code can be generated by applying code generator.

V. RELATED WORK

In [15] UML is used to model Web services, by presenting Web services as a class diagram. Their main contributions were conversion rules between UML and web services described by WSDL and XML Schema. However since UML is a general modeling language, in some cases it becoming less user friendly, for example, service-based system that are use event-based interactions.

Service Component Architecture (SCA) [12] is a set of specifications which describe a model for building software applications. An SCA component (a service model) has interfaces and references where we can use to build service-based systems. An SCA component consists of a configured implementation, where an implementation is the piece of program code implementing business functions. SCA emphasizes the decoupling of service implementation and of service assembly from the details of infrastructure capabilities and from the details of the access methods used to invoke services. With this, a service-based system is defined as interactions of SCA components using wiring concept. But unfortunately, SCA does not explicitly specify events and therefore it is difficult to model service-based systems that use event-based interactions.

SoaML [5] is a specification for the UML Profile and Metamodel for Services. SoaML is a standard extension to UML 2 that is meant to facilitate services modeling. With SoaML, a service is presented using UML profiles. However, events which are fundamental properties of service-based system that use event-based style of interactions is not described in the SoaML specification.

We adapt a CCM to model a basic service and present it as a building block with ports. We consider that a building block is a kind of activities encapsulation which can be access though its ports. Activities encapsulation [16] are potential be used for an incremental development of software system. To model a composite service we connect these building blocks using its provided ports.

This work is strongly related to Arctis [13] since our target is presenting existing services (described in service descriptions) in Arctis. However, our concept of building block is different since building block in Arctis is not specified only for presenting services, especially with the use of ports. However, the use of activity diagrams to define interactions between building blocks is the same concept. We are agree that with this approaches, a small, verified, and validated encapsulated activity diagram (a building block) can be used to build incrementally a larger building block diagram presenting a complex software system.

For the code generation, we directly interpret the activity diagram into code. The potential code generation from activity diagram was studied in [17]. With regard to their study, a building block is an entity that executes an external action. For a tool, Enterprise Architect from Sparx Systems is an example of modeling tools that support for code

generation from activity diagram.

VI. Concluding Remarks and Future Work

Software applications tend to be more component-based. Being component-based, two things are required: a way to create components and a mechanism for describing the interactions of those components. Furthermore, with model-driven development approaches we need to visually present those components and their interactions graphically.

This paper proposes to adapt the CCM for presenting services, and to present a service as a building block. The internal service behavior is described using an activity diagram that can be accessed via ports. These ports are used to interact with other building blocks. The interactions between building blocks itself are defined using activity nodes (activity diagrams) on one single view of model.

Each building block refers to an implementation class that acts as a proxy to the actual services. It means that our proposal is trying to use model as a visualization of software unit (i.e. services) to support model-driven development of service-based applications.

Currently we work only on the XML-based service descriptions. We will work further on services that are implemented in programming languages such as CORBA services.

References

[1] OMG, "Model Driven Architecture Guide, Version 1.0.1," *Object Management Group*, Jun-2003.

[2] OMG, "Unified Modeling Language Specification, version 2.2," *Object Management Group*, 2009.

[3] OMG, CORBA Component Model Specification, v4.0. Object Management Group.

[4] T. Allweyer, BPMN - Business Process Modeling Notation. BoD, 2009.

[5] OMG, "Service oriented architecture Modeling Language (SoaML): Specification for the UML Profile and Metamodel for Services (UPMS)," *Object Management Group*, 2009.

[6] S. Konno, "Cyberlink for Java Programming guide v.1.3," 2005.

[7] E. Zeeb, A. Bobek, and H. Bohn, et. al, "WS4D: SOA-Toolkits making embedded systems ready for Web Services," in *Proceedings of Second International Workshop on Open Source Software and Product Lines*, pp. 33–42, 2007.

[8] J. Goodwill, Apache Axis Live: A Web Services Tutorial. Sourcebeat, 2004.

[9] D. McIlroy, "Mass-Produced Software Components," in *Proceedings of the 1st International Conference on Software Engineering*, pp. 98, 88, 1968.

[10] O. Dahl, B. Myhrhaug, and K. Nygaard, "Some features of the SIMULA 67 language," in *Proceedings of the second conference on Applications of simulations*, pp. 29–31, 1968.

[11] C. Peltz, "Web Services Orchestration and Choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.

[12] Service Component Architecture. Nov. 1, 2006. Service Component Architecture. [Online]. Available: http://www.ibm.com/developerworks/library/specification/ws-sca/

[13] F. A. Kraemer, "Arctis and Ramses: Tool Suites for Rapid Service Engineering," in *Proceedings of NIK 2007 (Norsk informatikkonferanse)*, Oslo, Norway, 2007.

[14] M. Jeronimo and J. Weast, UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play. Intel Press, 2003.

[15] R. Grønmo, D. Skogan, I. Solheim, and J. Oldevik, "Model-Driven Web Services Development," in *e-Technology, e-Commerce, and e-Services, IEEE International Conference on*, vol. 0, pp. 42–45, 2004.

[16] F. Kræmer and P. Herrmann, "Automated Encapsulation of UML Activities for Incremental Development and Verification," in *Model Driven Engineering Languages and Systems*, vol. 5795, Springer Berlin / Heidelberg, 2009, pp. 571-585.

[17] A. Bhattacharjee and R. Shyamasundar, "Validated Code Generation for Activity Diagrams," in *Distributed Computing and Internet Technology*, vol. 3816, Springer Berlin / Heidelberg, 2005, pp. 508-521.