# AGENT-ORIENTED MODELING AND DESIGN METHODS FOR E-SERVICES/WEB SERVICES

Milan VARGA, Martin HUŇADY, Zdeněk HAVLICE
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic,
E-mail: mvarga@gmail.com, hunady@netkosice.sk, zdenek.havlice@tuke.sk

**SUMMARY**

*The component modeling offers the potential to assemble applications much more rapidly than ever before. Agents can be viewed as specialized distributed components, offering greater flexibility than traditional components when developing certain applications. Different kinds of agents have different amounts of personality, mobility, interaction, collaboration, persistence and intelligence. Web services and service-based architectures are quickly becoming the standard development model for software applications. Agent-oriented software engineering techniques are becoming more popular which offer a new perspective to developing applications. Agents are an excellent technology for implementing web services or e-services. Web services and agents integration brings benefits both technologies.*

***Keywords:*** *web service, agent, integration, MDA, agent planning, BPEL, DSL, workflow*

## 1. INTRODUCTION

Today's companies must be agile in adapting their business applications to the changing market dynamics if they want to stay competitive. Selecting the right technology for automating and integrating business processes is always a challenging task. Research in distributed AI is focused towards building a world comprised of individual agents where no one particular agent is omnipotent but the system as a whole is intelligent. These agents are separate atomic entities that communicate and coordinate with each other to achieve goals that are beyond the capabilities of any one agent. Agent technology could be used in integration scenario. In this context, Web services are conceived as an essential component for promoting interoperability of business processes and software agents as a key enabling technology for such processes to be dynamic.

### 1.1. Disadvantages of traditional component model

The most common component model in the Windows environment is COM. If they are designed correctly, COM components can be re-used by the other components and applications easily. In the Java world, CORBA is common solution, but components can also exist as Java servlets or other variations. In the modern distributed application environment, however, there are problems with the component-based architecture approach [3]. Components written in different programming languages are not so compatible one might like. But even if the cross-language problems are fixed, there is a bigger problem: It is very difficult for components to be shared across heterogeneous platforms. That is, calling a COM object from a Java program, or a CORBA object from a Visual Basic application. Cross-platform interoperability is not easy with the component models we have been using for the past decade. Calling a foreign object from beyond the firewall is not possible. How to fix those handicaps?

We need two things [5]:

1. First, every component-based application needs to speak the same language.
2. Second, instead of thinking in terms of processes, components, and data, we need to think in terms of services.

## 2. WEB SERVICES

Web service (WS) refers to distributed or virtual applications or processes that use the Internet to link activities or software components [4]. For example, a travel Web site that takes a reservation from a customer, and then sends a message to a hotel

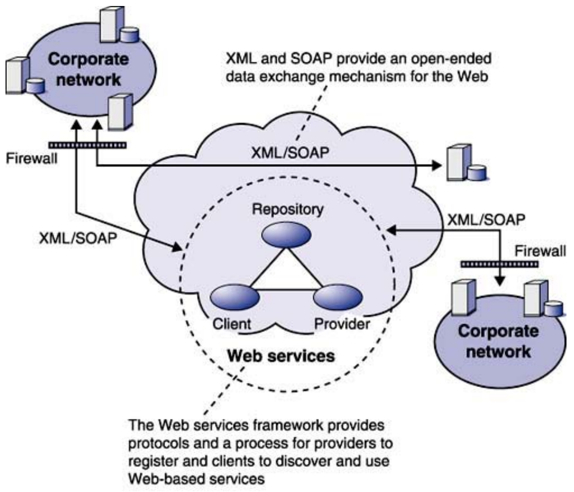
Corporate network
XML and SOAP provide an open-ended data exchange mechanism for the Web
Firewall
XML/SOAP
XML/SOAP
Repository
XML/SOAP
Firewall
Client
Provider
Web services
Corporate network
The Web services framework provides protocols and a process for providers to register and clients to discover and use Web-based services

**Fig. 1** Web service framework [4]

*This work was supported by VEGA 1/2176/05 - Technologies for Agent-based and Component-based Distributed Systems Lifecycle Support.*

application, accessed via the Web, to determine if a room is available, books it, and tells the customer he or she has a reservation is a WS application. WS provides a framework for communication across the Web. It is both a set of protocols and a process for discovery and connection (Figure 1).

From the perspective of agents, Web services are simply programmatic entities that can be called upon to perform an advertised and typically unitary function. To consumers of Web services, agents can form a powerful means of indirection by masking the Web service for purposes of redirection, aggregation, integration or administration.

## 3. AGENTS

Agent technology is a paradigm for conceptualizing, designing, and implementing software systems. This technology can be used for creating software operable in environments that are distributed and open. A software agent, whether intelligent or not, is a program that performs a specific task on behalf of a user, independently or with little guidance. An intelligent agent performs, reactively and/or pro-actively, interactive tasks tailored to a user's needs without humans or other agents telling it what to do. To accomplish these tasks, it should possess the following general characteristics of [1]:

- Independence
- Learning
- Cooperation
- Reasoning
- Intelligence

For intelligent software agents, knowledge representation is a crucial issue. As any AI application, what agent is expected to do, and in what domain, will have a significant impact on the type of knowledge representation that should be used. If agent has to solve a limited number of situations its intelligence could be embedded into procedural program code. If agent has to build or use sophisticated models of the problem domain and solve problems at different levels of abstraction, then frames or semantic nets could be used. If the agent has to answer questions or generate new facts from the existing data, then predicate logic or if-then rules should be considered. The amount of intelligence, in terms of both domain knowledge and power of the reasoning algorithms, required by agent is related to the degree of autonomy and, to a lesser extent, mobility it has. If agents have to deal with a wide range of situations, then it needs a broad knowledge base and a flexible inference engine. If agent is mobile, there may be a premium on having a small, compact knowledge representation and a light -weight reasoning system in terms of code size. If the agent is long-lived and will perform similar tasks many times during its lifetime, then learning can be used to improve its performance. [2]

### 3.1. Agent planning

Planning is essential for agents that act in an environment. To solve a goal intelligently, an agent needs to think about what it will do now and in the future. For reaching the agent goals agent use a set of plans. Those plans could be represented in a process of analyze and design in the form of workflow charts (Figure 2). Plan provides the description of the behavior and interactions of an agent instance relative to its partners and resources through communication interface. Plan execution starts only when the activation condition is met. Plans are executed in priority order. The semantics of a plan process definition can be described in rather simple terms. The basic building block could be an activity (sometimes called task or action). Usually, an activity can send a message to another agent, wait for an incoming message, or execute a specific function internally. Activities can be connected in serial fashion, or multiple activities can be executed in parallel using a fork and join construct. A fork allows multiple activities to execute at the same time. A join synchronizes multiple parallel threads of execution back into a single thread. Execution after a join can continue only if all parallel threads have completed their respective activities. The process template also must be able to specify a branch, or decision point, so that the path of execution can change based on the content of a message field. The formal representation for agent plan could be described with grammar. Suitable grammar is BPEL4WS which graphical notation provides a way of visualizing the often-complex workflows based on Web service.

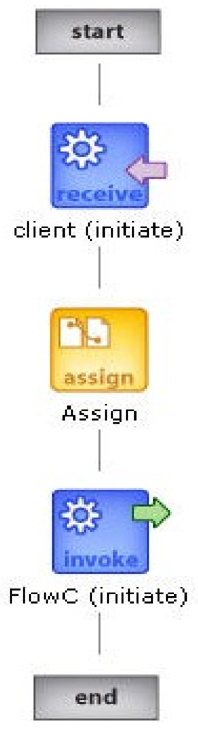

**Fig. 2** Agent plan example

### 3.2. BPEL4WS

BPEL4WS notation can be interpreted as a formal definition of grammars, which provides the description of the behavior and interactions of a process instance relative to its partners and resources through Web service interfaces. BPEL4WS provides a standard XML language for expressing business processes consisting of functions defined through Web services interfaces. BPEL4WS has both design and runtime uses. At design time, development or modeling tools can use, import, or export BPEL4WS to allow business analysts to specify processes, and developers to refine them and bind process steps to specific service implementations (Figure 2). The runtime choreography or workflow engine can use BPEL4WS to control the execution of process, and invoke the services required to implement them. Within the context of business integration, BPEL4WS addresses a definite requirement for an integration framework that enables the meet-in-the-middle between the business people and the IT world. A business analyst can use its modeling tool to choreograph business functions into a logical process model. The business process can then be exported and translated into BPEL4WS, which IT can apply it into a service model using Web services. The BPEL programming language supports:

- A property-based message correlation mechanism
- XML and WSDL typed variables
- An extensible language plug-in model to allow writing expressions and queries in multiple languages: BPEL supports Xpath 1.0 by default
- Structured-programming constructs including if-then-elseif-else, while, sequence (to enable executing commands in order) and flow (to enable executing commands in parallel)
- A scoping system to allow the encapsulation of logic with local variables, fault-handlers, compensation-handlers and event-handlers
- Serialized scopes to control concurrent access to variables

## 3.3. BPEL4WS Loan Flow Demo Orchestration

Let's imagine that you have implemented and deployed Loan Flow scenario. Here are the instructions for initiating a test instance of the flow through the BPEL (Figure 3). Behind the scenes, an XML Loan Application document is generated and posted to the BPEL Orchestration Server. The server consumes the message, initiates a flow instance and starts processing it asynchronously. As you can see, the flow invokes the credit rating service and then initiates in parallel a conversation with 2 asynchronous loan processor services. United Loan has already called back with a loan offer but the flow is still waiting for the asynchronous callback from StarLoan As you can see, the <receive> activity has been activated and it is waiting for the callback message from StarLoan. Once the response message arrives, the <receive> activity will consume it and the orchestration server will resume processing the BPEL scenario.

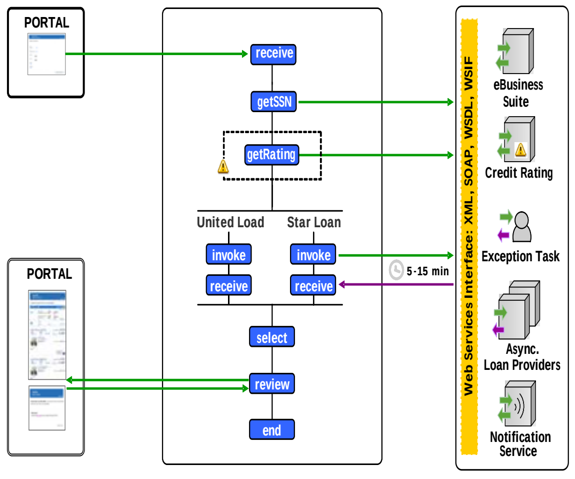

**Fig. 3** BPEL Loan flow sample

## 3.4. Modeling Agent Plan with the Model Driven Architecture

An agent plan can be represented as Platform Independent Model (PIM) [8]. Plan is a set of actions used by an agent, this set can be imported into a Model Driven Architecture (MDA) -based UML modeling tool and viewed as a group. Zooming out to show the least detail about each action allows us to view and model how they do, or could, work together. Modeling at this high level is important advantage. It allows us to focus on business functionality and behavior without worrying about technical aspects. Existing DSL framework has abilities for PIM model graphical representation and creation. This PIM model is automatically processed and translated by software agent to PSM model. In our article PSM model is represented in BPEL language (Figure 2). Transformation from PSM to Code is done internally in most of workflow frameworks or engines, which are supporting BPEL language. MDA advantages for agent plan modeling are described in Tab. 1. Next part of article is focused on explanation of MDA development life cycle.

## 4. MODEL DRIVEN ARCHITECTURE AND DOMAIN SPECIFIC LANGUAGES

Model Driven Architecture (MDA) is a framework for software development defined by the Object Management Group (OMG). Applications can be made independent of the infrastructures they use. MDA was designed to help organizations rapidly adopt new technologies and concepts without necessitating a rewrite of their entire systems. MDA is touted as the biggest shift in software development since the move from assembler machine code to the first high-level languages. Key to MDA is the importance of models in the software development process. Within MDA the development process is driven by the activity of modeling your software system. Unified Modeling Language (UML) is the heart of MDA. MDA uses UML diagrams to design and describe software applications. Using models as an essential part of the development process is now recognized as a best practice, known generally as Model Driven Development (MDD).

### 4.1. The MDA Development Life Cycle

The MDA development life cycle, which is shown in Figure 4 does not look very different from the traditional life cycle. The same phases are identified. One of the major differences lies in the nature of the artifacts that are created during the development process. The artifacts are formal models, i.e., models that can be understood by computers. The following three models are at the core of the MDA.

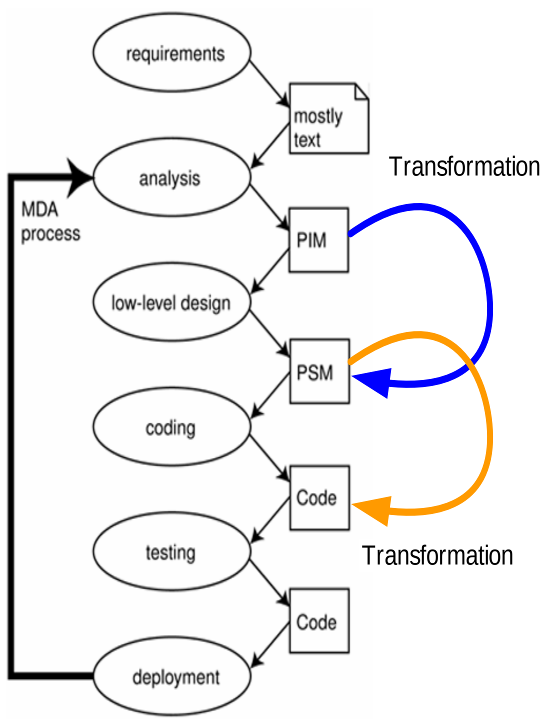

**Fig. 4** MDA software development life cycle

### 4.2. Platform Independent Model

The first model that MDA defines is a model with a high level of abstraction that is independent of any implementation technology. This is called a Platform Independent Model (PIM). A PIM describes a software system that supports some business. Within a PIM, the system is modeled from the viewpoint of how it best supports the business. Whether a system will be finally implemented plays no role in a PIM.

### 4.3. Platform Specific Model

In the next step, the PIM is transformed into one or more Platform Specific Models (PSMs). A PSM is tailored to specify your system in terms of the implementation constructs that are available in one specific implementation technology. A PIM is transformed into one or more PSMs. For each specific technology platform a separate PSM is generated. Most of the systems today span several technologies therefore it is common to have many PSMs with one PIM.

### 4.4. Code

The final step in the development is the transformation of each PSM to code. Because a PSM fits its technology rather closely, this transformation is relatively straightforward. The MDA defines the PIM, PSM, and code, and also defines how these relate to each other.

A PIM should be created, and then transformed into one or more PSMs, which then are transformed into code. The most complex step in the MDA development process is the one in which a PIM is transformed into one or more PSMs.

### 4.5. Domain Specific Language

A domain-specific language (DSL) is a programming language tailored specifically to an application domain [7]: rather than being general purpose it captures precisely the domain's semantics. A DSL-based development methodology addresses the need for increasing domain specialization in the software engineering field. Examples of DSLs include lex and yacc used for program lexical analysis and parsing, HTML used for document mark-up, and VHDL used for electronic hardware descriptions. Domain-specific languages allow the concise description of an application's logic reducing the semantic distance between the problem and the program. In May 2005 Microsoft released Microsoft Tools for Domain-Specific Languages [6] (Figure 5) that is a suite of tools for creating, editing, visualizing, and using domain-specific data for automating the enterprise software development process. These new tools are part of a vision for realizing software factories.

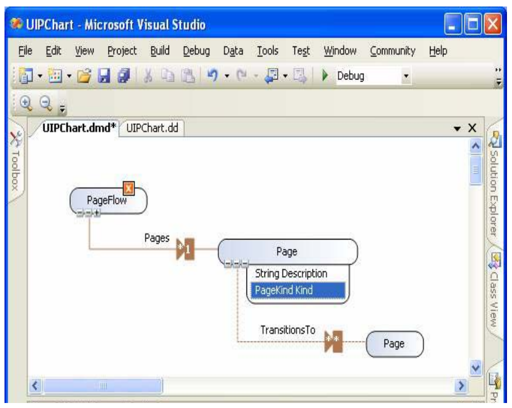
UIPChart - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
UIPChart.dmd* UIPChart.dd
Toolbox
Solution Explorer
Class View
PageFlow
Pages
Page
String Description
PageKind Kind
TransitionsTo
Page

**Fig. 5** Microsoft DSL tool [6]

## 5. CONCLUSION

The idea was to find representation for agent plan that describes agent activities and their interdependencies. This representation could be formally defined with BPEL4WS grammar. Integrating Web services and software agents bring about the immediate benefits of connecting application domains hosting one or the other technology. A Web service should be able to invoke an agent service and vice versa.

Future research and development work will be focused on new paradigms of information systems integration based on AI components.

| **Business driver** | **MDA promise** |
|---|---|
| Reduced risk of lock-in to specific technologies | Ensures that rapid changes in technology frameworks do not render applications useless |
| Preservation of investment in application development | As above, ensures that rapid changes in technology frameworks do not render applications useless |
| Increased productivity | Through promotion of code reuse and code generation, repetitive coding is eliminated from projects |
| Increased business agility | Ability to respond to changing business needs by changing the application model to fit changes at requirements level |
| Reduced development and maintenance costs and reduced time-to-market | ▪ Simplifies the task of maintaining the software during a long production lifetime by being able to autoport the application to new platforms<br>▪ Maintenance activities rendered more cost-effective<br>▪ Manpower overhead required to create software reduced<br>▪ No need to code complex infrastructure plumbing<br>▪ Automation of repetitive development tasks |
| Improved application quality | ▪ MDA recognizes the need to improve source code: the less code written, the less liability assumed<br>▪ Tested software blueprints, industry-standard patterns, can be automatically applied. This prevents the deviations from architecture and design guidelines that are the primary source of scalability, performance, and availability problems |
| Software that meets business needs | ▪ Business-focused approach to software development, with the ability to quickly react to changing business conditions<br>▪ Tested application blueprints help provide industry standard software |

**Tab. 1** Business drivers for Agent oriented applications based on MDA

## REFERENCES

[1] R. Kalaoka, A. Whinston: The Frontiers of Electronic Commerce, Addison-Wesley, 1996

[2] Bigus, J.: Constructing Intelligent Agents with Java. Willey Computer Publishing, Canada, 1997.

[3] Greenfield J., Keith S.: Software Factories - Assembling Applications with Patterns, Models, Frameworks, and Tools. John Wiley & Sons, 2004.

[4] Douglas K. Barry: Web Services And Service-Oriented Architecture. Morgan Kaufmann Publishers, 2003.

[5] FoodMovers: Building Distributed Applications using Microsoft Visual Studio .NET http://www.msdn.microsoft.com/library/default.asp, 2003

[6] Domain-Specific Language (DSL) Tools http://lab.msdn.microsoft.com/teamsystem/workshop/dsltools/default.aspx, 2005

[7] Notable Design Patterns for Domain-Specific Languages http://www.spinellis.gr/pubs/jrnl/2000-JSS-DSLPatterns/html/dslpat.html

[8] Kleppe A., Warmer J., Wim B.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison Wesley, 2003.

## BIOGRAPHIES

**Martin Huňady** was born on 03.06.1980. In 2003 he graduated (MSc.) with distinction at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He study his PhD. in the field of information systems and Technologies; his thesis title is "Modeling and prototyping dynamical system properties".

**Milan Varga** was born on 30.11.1979. In 2004 he graduated (MSc.) with distinction at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He study his PhD. in the field of information systems and Technologies; his thesis title is "Integration of knowledge and information systems".

**Zdeněk Havlice** was born on 14. 02.1958. In 1982 he graduated (MSc.) with honors at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He defended his PhD. in the field of visual programming and user interface design in 1991; his thesis title was: "Design of User Interface for Dialogue Systems". Since 1999 he is working as an associated professor at the Department of Computers and Informatics. His scientific research is focusing on the area of special languages, compilers, CASE systems, software methodologies, methods and tools.