

## TOWARDS A SERVICE-ORIENTED DEVELOPMENT METHODOLOGY

**Alexander Ivanyukovich**

**G. R. Gangadharan**

**Vincenzo D'Andrea**

**Maurizio Marchese**

Department of Information and Communication Technology, University of Trento,  
38100 Trento, Italy

*Leveraging a service-oriented paradigm would significantly affect the way people build software systems. However, to achieve this ambitious vision a solid software development methodology should be in place, comprising specific, service-context patterns as well as appropriate supporting tools which altogether integrate methods and best practices into a stable development environment. This paper presents a structured approach to analyzing software development methodologies in light of the specific features of service-oriented applications. The proposed approach is based on a critical assessment of existing software development methodologies along three methodological dimensions, namely managing change in software development, specifying the software development process and targeting the stakeholder goals. For each of this dimension we first identify suitable software methodologies capable to meet the specific challenges and then discuss their contribution towards the definition of a service-oriented development methodology.*

**Keywords:** Service Oriented Architectures, Service Oriented Software Engineering.

### 1. Introduction

During the last decade the idea of software composition and refinement instead of software development from scratch has emerged. Nowadays this trend is elaborated to the platform-independent, distributed and standardized services paradigm. In such a paradigm, services embody coarse-grained, loosely coupled entities that can be described, published, discovered and invoked in a distributed environment regardless of their platform and location. Each service represents self-contained and self-defined functionality usually supported by back-end systems (Papazoglou *et al.*, 2004).

Service-Oriented Architectures (SOAs) are evolving to address the emerging aspects of real world applications. They mainly target the market of enterprise information systems, where boundaries (internals and externals) are not fixed and are subject to constant change and never-ending improvements to gain competitive advantages. The basic SOA build on the publish-find-bind scheme and can be utilized as an architectural skeleton in most service-based projects. However, this scheme does not address many aspects of real-world applications like composition, management, security, performance, reliability – to name a few. More complex architectures have been proposed to fulfill this gap (Papazoglou *et al.*, 2004; Ivanyukovich *et al.*, 2005; Lazovik *et al.*, 2004; Aggarwal *et al.*, 2004; Huang *et al.*, 2004). However, the direct application of SOA in the business-to-business world exposed serious flaws on methodological levels. Although some foundation concepts of service-oriented design

have already been addressed (Kleppe *et al.*, 2003; OASIS, 2004), a comprehensive methodological approach is still missing.

This paper presents a structured approach to analyzing software development methodologies in light of the specific features of service-oriented applications. The proposed approach is based on a critical assessment of existing software development methodologies along three methodological dimensions, namely managing change in software development, specifying the software development process and targeting stakeholders goals. For each of this dimension we have identified suitable software development methodologies capable to meet the specific challenges. In the next section we are describing briefly the main aspects of each selected dimension and related development methodologies.

## **2. Managing the Change: the XP Approach**

The agile approach of software development methodology buds from the reality of today's volatile markets. The spiral model (Boehm, 1988) has influenced the modern day concept of agile. There is no essential difference between the agile approach and spiral model. However, protracted planning/analysis phases and a corroborated emphasis on copious documentation have prevented projects from being truly agile (David *et al.*, 2003) in using spiral techniques. The emergence of agile software processes attempts to deal with the issues introduced by rapidly changing and unpredictable markets. The agile approach recognizes that software applications will evolve as the business context varies. The aim of the agile approaches is to allow an organization to deliver and to change quickly (Highsmith *et al.*, 2001).

Extreme Programming (XP), the most perceptible of the agile processes, is a lightweight discipline of software development with values of communication, simplicity, feedback, and courage (Beck, 1999) through a set of defined practices. Recent experiences suggest the extensibility of agile approaches for scalable and distributed environments.

### **2.1. Adopting and Adapting the XP for Service Oriented Development Methodology**

"It is not the strongest of the species that survives, nor the most intelligent - it is the one that is most adaptable to change," said Charles Darwin. Embracing change is the indispensable concept of the service oriented paradigm as well as agile approaches. In the world of services that interoperate and interact with each other using Web-based standards, services should be developed with change and integration in mind; hence SOA appears as an adept candidate for the XP approach.

The iterative and incremental development (Craig *et al.*, 2003) paradigm is the common denominator of agile processes for confronting a vague process showing the changing requirements (Favaro, 2002). Requirements may be introduced, modified or removed in successive iterations. This approach would retain tractability in requirements evolution, which is a very important problem for any complex service-oriented system, while keeping costs down since we do not assay to implement the entire universe from the beginning. A service can be viewed as an aggregation of one or many features depending upon the complexity. Based on the planning game, the development progresses further with iterations.

XP emphasizes designing the simplest plausible solution (Wake, 2000) that can be implemented at the moment, eliminating superfluous complexity. Specifying the service interface and specifying operation parameters should be definitively implemented based on the requirement when a specific functionality emerges with no future thinking. The more accurate service interface results in a more accurate resemblance to the real world process.

The nucleus of XP is with writing little tests first (Jeffries *et al.*, 2000), and with always expressing the intention, not algorithm, in the code written. Service oriented computation is evolving towards combining web services to create higher level inter-organizational business processes dynamically (Peltz, 2003). Combining various fine-grained web services into larger coarse-grained services

becomes a complex orchestration process. If every individual web service encompasses its own automated test, then composing web services could also include the composition of tests. This obviates the complexity in rolling out new versions of web services.

The initial endeavor on mapping the aspects of XP methodology to the service-oriented framework heralds the following reflections:

- Services could be related to the idea of features describing briefly the behavior of the system. Thus this may be an apposite way to approach the development of web services, keeping strict adherence to user requirements, while reining in costs.
- Developing web services incrementally would augment for a complete, functioning system rapidly riveting on simplicity.
- Adopting test driven strategy seems a provoking catalyst for complex and dynamic orchestration of web services by automatic coordination of the appropriate tests on all levels of granularity.

The XP practices (Beck, 1999) seem to be motivating in the domain of services development. A primary evaluation is as follows:

<b>XP Practice</b>	<b>Impacts in Services Development</b>
Planning game	Viewing services as a set of features
Frequent Releases	Definitively an advantage for SOA, with immediate feedback from customers
Simple Design	Defining service interfaces with operational parameters
Testing	Inherent for automating tests with services
Refactoring	For smooth evolution of design of services
Coding standards	Strict adherence will be useful for orchestration and mapping
Continuous integration	Required for service orchestration and choreography
On-site customer	May not have significant impact; but preferable
Metaphor Pair programming Collective ownership Forty hour week	We believe that these practices are not relevant for services development

In order to define a complete agile methodological approach, we are considering the definition of a number of postulates. Predictions will converge as postulates when:

- multiple case studies are demonstrated with sets of user stories and real incremental implementations for web services by XP approach
- few practices of XP approach are adapted for practicing in service oriented development with scalable and distributed environment
- the roles in the domain of web services are identified and integrated into an agile approach.

### **3. Specifying the Process: the RUP Approach**

Rational Unified Process (RUP) is one of de facto world-wide accepted standards in the software development industry. Iterative software development, requirements management, quality assurance and other practices (Kruchten, 2000) are now integral parts of the process behind any software project. They have proven to be useful for numerous projects and now can be found as the best practices part of RUP. RUP has its own software development lifecycle model comprising of inception, elaboration, construction and transition. However, lifecycle in RUP is different from the classical waterfall model

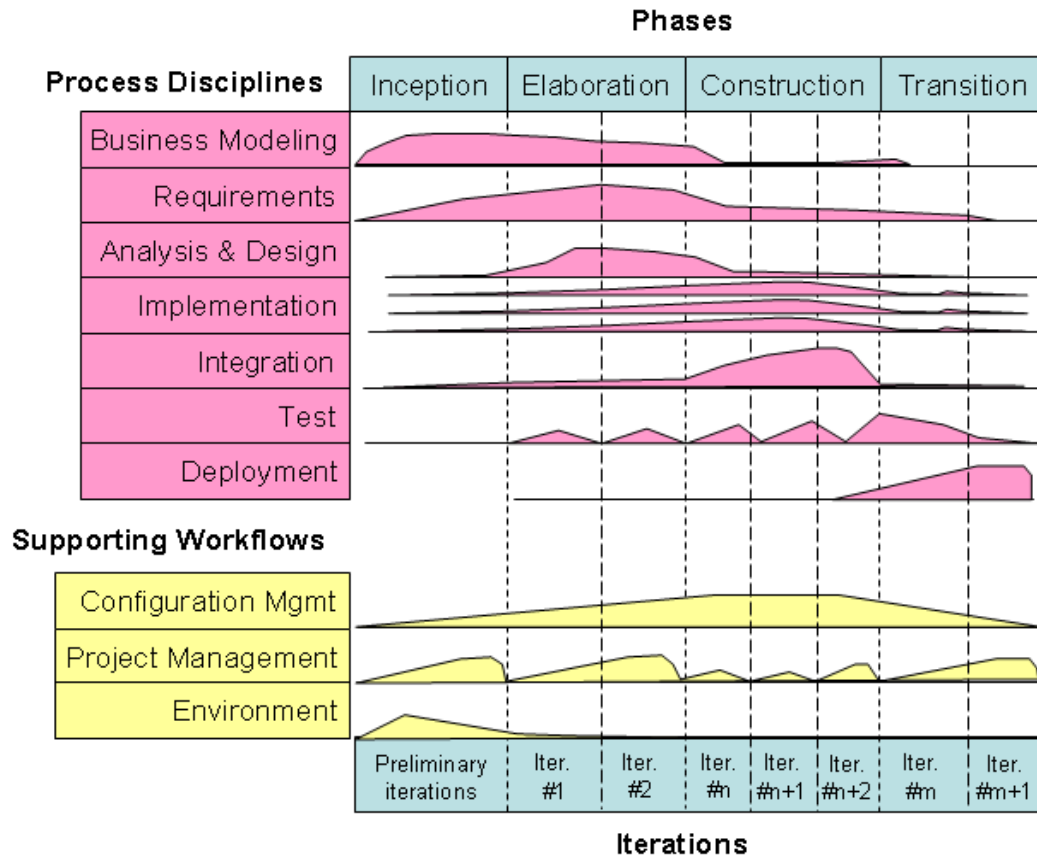
stages in a sense that it represents the major business decisions points rather than technical evolution milestones.

The primary application area of RUP is long-term projects. While RUP belongs to the traditional methodologies family, it covers all phases of the development process in details. Originally it was introduced to the component-oriented design concept; however it does not have strict limitations that will prevent its usage within other conceptual frameworks. Although RUP states that it should be tailored to a particular project, it specifies strict rules for roles, responsibilities, and projects schedule planning thus contributing to process stability (Rational, 2002).

### **3.1. Adopting and Adapting the RUP for Service Oriented Development Methodology**

Service-oriented concept defines a new level of architectural principles that is largely analogous to the existing object- and component-oriented software design concepts. Nevertheless the aspect of loosely-coupled services engenders considerable differences in the way SOA applications are built compared with traditional object- and component-based software. Practical methods currently in use for developing SOA and component-based systems are the same. However, delivering SOA applications developed under RUP requires customizing the process to address SOA applications planning. These differences have an impact on the way RUP can be applied for developing and delivering SOA applications. The software development process should resolve the following issues:

- The loose-coupling of services allows planning and developing each particular service in the way that is best suitable for it. Thus overall service development life-cycle may be supported in RUP methodology via parallel per-service sub-projects that should converge for final integration and testing in the particular time sequence.
- Specific attention should be paid on the methodologies and tools and technologies part during inception and elaboration phases for the integration discipline. Ease of services integration and error-free operation largely depends on the inter-service operations planning quality.
- Behavioural constraints cannot be formally specified within interfaces description – interface description does not support specification of methods execution sequence (Brown *et al.*, 2002). This is a subsequent heritage left from components-based design.
- Planning for the ad-hoc and potentially asynchronous communication affects overall system design. This is what makes the difference with the tightly-coupled objects/components systems. Although a particular part of the whole project may utilize synchronous communication mechanisms, asynchronous processing should be considered as a principle means to minimize time to process data.
- The coarse-grained essence and loosely-coupled nature of services allow us to claim the necessity of detaching the integration step of the implementation workflow into a separate workflow unit.
- Enterprise-scale applications integration constraints are not resolved within a component-based design scheme (Brown *et al.*, 2002).
- A service-oriented paradigm mitigates the ongoing change in business requirements with extra flexibility of the suggested architecture; on the other hand RUP pursues early change accommodation.



**Fig. 1 Dynamic aspects of RUP with respect to SOA**

From the service oriented perspective we are pursuing the definition and addition of two major customization features to RUP, namely (see also Figure 1):

- the addition of a new process discipline called “Integration” that addresses convergence of individual services into a well-functioning application. This discipline is especially active during the elaboration phase when software architecture should be defined. This discipline is critical to avoid unnecessary rework late in the construction phase, when major modification should be needed to integrate different services.
- a change in the way the Implementation discipline is perceived and planned. Implementation consists of rather individual sub-projects, each delivering a particular component to the target SOA application. It is critical that all projects finish in correct sequence during the Construction phase. In contrast to traditional RUP, Construction phase is characterized by increased Integration discipline activity towards the end and decreased activity in the Implementation. The rationale is that in order to fulfill the projects goals, a working SOA application, all services should be ready before final integration.

#### 4. Targeting the Goals: TROPOS and MAP Approaches

Application service provision calls for software systems having open, evolving architectures, operate robustly and exploit resources available in an open environment. To build such systems,

software engineers need mechanisms for communication, negotiation, and coordination between software components as well as appropriate methods to assure the fulfillments of stakeholders' goals. The family of multi-agent and goal-driven methodologies can be used for guidance and support in building such open software systems. In the following two sections, we will present and discuss two approaches particularly interesting, in our view, for supporting service oriented development methodology.

#### **4.1. TROPOS: an Agent Oriented View of Software Development**

Most software development methodologies put emphasis on the use of requirements for identifying the needs motivating the development of a software system. Requirements have an important role in the first stage of the software development life-cycle, and in some cases are later used as a contractual and legal reference for the validation of the implemented system.

TROPOS is a methodology founded on modeling user requirements, extending their role after the first stage of development (Giorgini *et al.*, 2005). The requirements model is used along all phases of software development as a baseline for development activities. Refinements and extensions keep the model aligned with changes in the requirements, promoting the use of requirements also for documenting the system being produced.

The core concept in TROPOS is the notion of "goal". Goals are the motivating factors in the development process, expressing reasons and motivations for system functionalities and features. In TROPOS the requirements are divided in two categories: early and late requirements, goals are extracted from the former and used for building the latter.

The early phase of requirement analysis is aimed at identifying domain stakeholders, their relationships and organizational structure, their dependencies. Goals are then built from this analysis. Starting from the goals, the late requirements then take into account the system being developed, and its relationships with users and stakeholders. The aim of the late requirements is to express how the system will be constructed and what will be the resulting features and functionalities.

In TROPOS, diagrams play an important role. Actor diagrams focus on the actors, their relationships and the relationships with functional and non functional requirements. Rationale diagrams are used to represent goals, sub-goals and their dependencies. In addition to the traditional use of logical predicates to express sub-goal satisfiability (such as AND, OR), TROPOS diagrams introduce additional operators expressing the positive or negative contribution of a goal to the satisfaction of another one. Goal satisfaction can be analyzed in TROPOS by means of tools that propagate goal satisfiability or conflicts between goals, using qualitative or quantitative reasoning (Giorgini *et al.*, 2005).

#### **4.2. Adopting and Adapting TROPOS for Service Oriented Development Methodology**

While the main area of application of TROPOS is in agent oriented systems, the methodology was also proposed in connection to service oriented computing (Aiello *et al.*, 2004). Goals and subgoals in a service oriented architecture are connected to the operations performed by services; in other words a service is instrumental in fulfilling a goal or subgoal, and composition of services are connected to first level goals. TROPOS methodology is then used for the formal validation of a goal that in turn corresponds to the validation of the service composition with respect to goal satisfiability.

An important aspect in the approach proposed by (Aiello *et al.*, 2004) is the validation of goals in terms of non-functional as well as functional requirements. Non-functional aspects of services are related to contractual issues, legal obligations, limitations, quality of service, performances and so on. The ability to model a non functional requirement is peculiar to the TROPOS approach and is related to the activities in the early requirement phase, when the focus of attention is on modeling organizational and relational issues.

The application of TROPOS to the development of service oriented systems is then quite promising. First of all, it provides a way to model and maintain a system of goals for the overall service architecture. In addition, this methodology allows modeling the non-functional aspects of services, filling a gap in current approaches.

#### 4.3. MAP: a Multi-model View of Process Modeling

The representation formalism proposed in MAP (Rolland *et al.*, 1999) represents a different goal-driven approach to the software development process and more specifically to requirements software engineering. MAP is based on a multi-model view of process modeling which supports dynamicity. Briefly, the approach builds on the notion of a labeled graph of “intentions” and “strategies” called a “map” as well as its associated “guidelines”. In (Rolland *et al.*, 2000) the proponents of the approach specify:

- an “intention” as a goal to be achieved by the performance of the process;
- a “strategy” as an approach, a manner to achieve an intention;
- a “guideline” as ‘a set of indications on how to proceed to achieve an objective or perform an activity’.

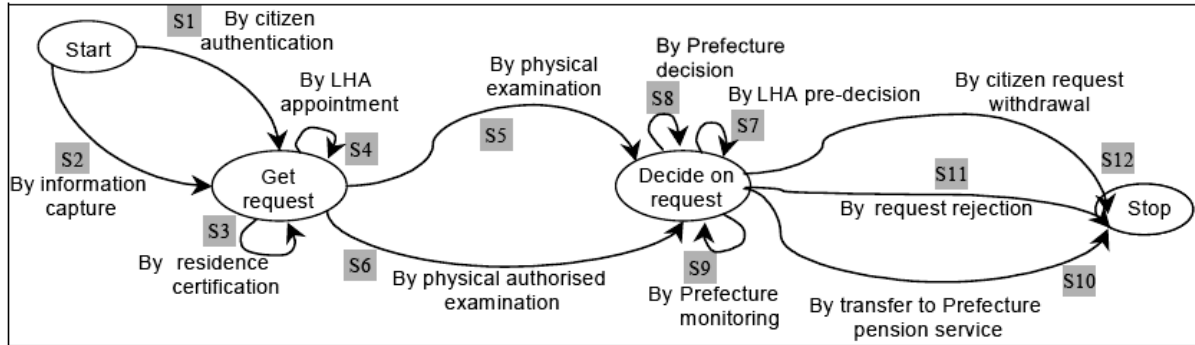
In this representation formalism, the key element is composed by a triplet  $\langle I_i, I_j, S_{ij} \rangle$ , called “section”, that represents a way to achieve the target intention  $I_i$  from the source intention  $I_j$  following the strategy  $S_{ij}$ . There are two distinct intentions called Start and Stop respectively that represent the intentions to start navigating in the map and to stop doing so.

Thus, it can be expected that there are a number of paths in the graph from Start to Stop. In a typical map (see Figure 2 as a specific example, from Kaabi *et al.*, 2004), none of the finite set of sections is recommended “a priori”. Instead the approach suggests a dynamic construction of the actual path by navigating in the map. In the practical use of the methodology, the next intention and strategy to achieve it are selected dynamically by the application engineer among the several possible ones offered by the map (thus, the multi-model view of process modeling advocated in the MAP approach). In such a case, “guidelines” that make available all choices open to handle a given situation are of great convenience. Therefore, a guideline embodies method knowledge to guide the application engineer in achieving an intention in a given situation.

Summarizing, the map is used as a navigational structure which supports the dynamic selection of the intention to be achieved next and the appropriate strategy to achieve it. The guidelines help in the realization of the selected intention.

MAP methodological approach is also based on the definition and use of “conceptual scenarios” that are described by maps. In particular, MAP recognizes three levels of abstraction called contextual, functional, and physical. The contextual level identifies the services that a system should provide to an organization and their rationale. The functional level focuses on the interactions between the system and its user to achieve the needed services. Finally, the physical level deals with the actual performance of the interactions.

An example of “functional” level of a conceptual scenario is represented in Figure 2, that models the functional level of a cooperative process of a virtual organization (composed by different organizations) whose goal is to “provide a pension to a disabled person”. The overall goal is subdivided in a number of sub-goals (“Get request” and “Decide on request”), and a number of strategies capable to achieve these goals are identified and included in the map.



**Fig. 2 Example of a functional conceptual scenario, from (Kaabi *et al.*, 2004)**

#### **4.4. Adopting and Adapting the MAP for Service Oriented Development Methodology**

The model proposed in MAP, expressed in goal/strategy terms, can be effectively used in a service oriented application development. Some first experiences have already been reported and discussed in (Kaabi *et al.*, 2004). In particular it can be used to assist:

- in the identification of the needed services, as well as eventual legacy services;
- in the discovery of the key playing agents in the overall process;
- in the correct distribution of services among the various agents;
- in the definition of the orchestration of services;
- in the enactment of the services execution in a model-driven manner.

The main characteristic of this approach relevant to services is its goal emphasis that is realized both in the modeling phase of the requirements and in the execution phase of the services that satisfy these requirements.

In this respect, the MAP approach supports the service developer in capturing first the objectives of a (business) process and then in deriving the associated functionalities (services) implied by the process. From this point of view, MAP approach is a top-down development approach capable to control that a given global objective is obtained and in discovering the services which support the achievement of this objective.

#### **5. Conclusions and Future Work**

Our current research on the definition of a complete service oriented methodology shows three important methodological dimensions relevant to the specific service domain, namely:

- managing the change in software development
- specifying the software development process and
- targeting the stakeholders' goals.

For each of them, we are analyzing and testing specific approaches with respect to services: XP for agile approach, RUP to specify the process and two goal-oriented approaches to target users' goals (TROPOS and MAP).

Each of the analyzed approaches have a contribution to the service domain, but also need to be adapted to the specific features, along the lines outlined in the previous sections. We believe that together, they will outline a comprehensive methodology for developing end-to-end service software products.

## 6. Acknowledgements

The authors would like to thank Aliaksei Yanchuk for useful discussions on the RUP approach, Marco Aiello and Paolo Giorgini on the TROPOS approach and Colette Rolland on the MAP approach. We are also grateful to Mike Papazoglou and Willem-Jan van den Heuvel for their comments.

## 7. References

- Aggarwal, R., Verma, K., Miller, J., and Milnor, W., 2004, "Constraint Driven Web Service Composition in METEOR-S," Proceedings of the 2004 IEEE International Conference on Services Computing, Shanghai, China, September 15-18, 2004, IEEE Computer Society, pp. 23-30.
- Aiello, M., and Giorgini, P., 2004, "Applying the TROPOS Methodology for Analysing Web Services Requirements and Reasoning about Qualities of Services," CEPIS Upgrade - The European journal of the informatics professional, Vol. 5, No. 4, pp. 20-26.
- Beck, K., 1999, "Extreme Programming Explained: Embrace Change," Addison-Wesley Professional, Reading.
- Boehm, B., 1988, "A Spiral Model of Software Development and Enhancement," IEEE Computer, vol. 21, no. 5, pp. 61-72.
- Brown, A., Johnston, S., and Kelly, K., 2002, "Using Service- Oriented Architecture and Component-Based Development to Build Web Service Applications," Rational Software Corporation, Santa Clara, California..
- Craig, L., and Victor, R.B., 2003, "Iterative and Incremental Development: A Brief History," IEEE Computer, Vol. 36, No. 6, pp. 47-56.
- David, C., Mikael, L., and Costa, P., 2003, "Agile Software Development - A DACS State-of-the-Art Report," DACS SOAR 11, Data and Analysis Center for Software, New York.
- Favaro, J., 2002, "Managing Requirements for Business Value," IEEE Software, Vol. 19, No. 2, pp. 15-17.
- Giorgini, P., Mylopoulos, J., and Sebastiani, R., 2005, "Goal- Oriented Requirements Analysis and Reasoning in the Tropos Methodology," Engineering Applications of Artificial Intelligence, Elsevier, Vol. 18, No. 2, pp. 159-171.
- Highsmith, J., and Cockburn, A., 2001, "Agile Software Development: The Business of Innovation," IEEE Computer, vol. 34, no. 9, pp. 120-122.
- Huang, Y., Kumaran, S., and Chung, J-Y., 2004, "A Service Management Framework for Service-Oriented Enterprises," Proceedings of the IEEE International Conference on E-Commerce Technology, San Diego, California, July 6-9, pp. 181-186.
- Ivanyukovich, A., and Marchese, M., 2005, "Service Oriented Architectures for Business Process Management Systems," Proceedings of IADIS International Conference Applied Computing, Algarve, Portugal, February 22-25, pp. 543-550.
- Jeffries, R., Anderson, A., and Hendrickson, C., 2000, "Extreme Programming Installed," Addison-Wesley Professional, Reading.
- Kaabi, R.S., Souveyet, C., and Rolland, C., 2004, "Eliciting Service Composition in a Goal Driven Manner," Proceedings of the International Conference on Service Oriented Computing 2004, New York, November 15-18, pp. 308-315.
- Kleppe, A., Warmer J., and Bast, W., 2003, "MDA Explained: The Model Driven Architecture™: Practice and Promise," Addison Wesley, Reading.
- Kruchten, P., 2000, "The Rational Unified Process An Introduction," Second Edition, Addison Wesley, Reading.
- Lazovik, A., Aiello, M., and Papazoglou, M., 2004, "Associating Assertions with Business Processes and Monitoring their execution," Proceedings of the International Conference on Service Oriented Computing 2004, New York, November 15-18, pp. 94-104.
- OASIS Business-Centric Methodology Technical Committee, 2004, "Business-Centric Methodology Specification v1.0," <http://www.oasis-open.org> (cited January 23, 2005).

Papazoglou, M. P. and Dubray, J., 2004, "A Survey of Web Service Technologies," Technical Report DIT-04-058, Department of Information and Communication Technology, University of Trento, <http://eprints.biblio.unitn.it/archive/00000586/4>.

Peltz, C., 2003, "Web Services Orchestration and Choreography" IEEE Computer, Vol. 36, No. 10, pp. 46-52.

Rational Software, 2002, "Rational Unified Process for Systems Engineering," A Rational Software White Paper (TP 165A), Rational Software Corporation, Santa Clara, California.

Rolland, C., and Prakash, N., 2000, "Bridging the Gap Between Organizational Needs and ERP Functionalities," Requirements Engineering, Vol. 5, No. 3, pp. 180-193.

Rolland, C., Prakash, N., and Benjamen, A., 1999, "Multi-Model View of Process Modelling," Requirements Engineering, Vol. 4, No. 4, pp. 169-187.

Wake, W.C., 2000, "Extreme Programming Explored," Addison-Wesley, Reading.