Chapter 24

A SERVICE ORIENTED APPROACH TO INTERORGANISATIONAL COOPERATION

Christian Zirpins, Winfried Lamersdorf, Giacomo Piccinelli

Abstract: Many E-business applications are based on increased cooperation between various organisational units and partners. System support for such applications can be provided using concepts from the area of *service oriented computing* – thus lifting inter-organisational integration to a higher level of effectiveness and efficiency. *E-services* provide means for modularisation of arbitrary organizational assets into components that can be dynamically offered, discovered, negotiated, accessed, and composed in an open application environment. Technically, E-services are software systems that are implemented on top of conventional information and communication technology. As an important step into that direction, *Web Services* have laid the foundation for interoperable communication between arbitrary systems. This paper introduces an approach to plan, build, and run such application-level services efficiently. Therefore, a fundamental notion of service, originating from distributed systems, is being extended by a specific concept of *cooperative interaction processes.* Accordingly, an application-level service model and corresponding service engineering mechanisms are proposed and realised as system software middleware based on OGSA Web Services and BPEL4WS processes.

Key words: E-Business, Inter-Organisational Integration, Cooperative Interaction Processes, Electronic Services, Workflow, Web-/GRID Service Architecture

1. INTRODUCTION: SERVICE ORIENTED DISTRIBUTED APPLICATIONS

Various application domains like electronic business, -government and -education face *recurrent cooperation scenarios* where a constant change of participants is a predominant characteristic. Typical examples are business-to-business integration problems (Medjahed et al., 2003) that focus on the dynamic relationship between one company and a set of frequently changing partners. Also, situations like flexible outsourcing of business functions or

dynamic supply chain management face similar recurring types of cooperation with interchangeable partners. For example, a company might contract out freight logistics to various carriers or forwarding agencies changing over time. The rationale for this kind of relationship is, on the provider-side, to expose new revenue streams (e.g. providing freight logistics on demand) and, on the customer-side, to seek for new efficiencies (e.g. outsourcing freight logistics if profitable) in a form that allows for constant optimization of partnership settings. Strategic planning of cooperation types, tactical preparation of cooperation settings, as well as operational control of functional cooperation are among the main challenges to be tackled here. In more advanced scenarios, the patterns of functional cooperation are often a subject of variation too, because different partners pose different operational requirements that have to be negotiated between the participants beforehand.

For example, a customer cooperates with various carriers that all move goods but impose different procedures of payment. Moreover, when broadening the scope, a party often faces multiple of such cooperative relationships that are in some cases mutually dependent. In order to preserve these dependencies, they have to be made explicit independently from individual partners. For example, a forwarding agency has to ensure that it can move goods of various individual customers by relying on alternating carriers under contract.

1.1 Extending the service notion

Henceforth, the notion of a *service* is used to refer to such recurring cooperation scenarios between changing autonomous participants. In order to substantiate this notion, one can benefit from former work in distributed information systems: Revisiting ODP concepts (ISO/IEC-JTC1/SC21, 1995), we distinguish the constant class of cooperation *(service type)* from changing cases of cooperation *(service instance).* Service instances can vary in the conditions of cooperation referred to as *service properties* (e.g. QoS) that arise from the characteristics of actual participants. Those participants are typed by *roles,* indicating expected cooperative behaviour within service relationships. Providers offer type and properties of instances they are willing to participate in. Clients observe offers of a specific type, choose a provider with respect to service properties and engage in service instances.

Specific interdependencies between services are often referred to as *service composition.* In this case, a participant relates *(composes)* services in which he acts as a provider *(composite services),* to services in which he acts as a client *(service components),* stating how characteristics of the composite service are put down to characteristics of service components. In terms of characteristics, services on application-level are more complex then those

found in classical distributed object systems. Apart from the 'semantic' reason (e.g. move goods), the 'syntactic' cooperation process (e.g. customer orders → carrier confirms and ships goods → customer pays) is among the predominant service characteristics. In particular, the focus here is on the interaction patterns, that is, the communication processes between roles.

The field of problems faced by organisations in terms of service participation can be structured into strategic, tactical and operational challenges. On the strategic level, exposing and expressing semantic and syntactic aspects of service types and their interdependencies requires expressive models and systematic design methodologies *(service modelling)* taking under account the (technological and conceptual) context of participants. For example, a forwarding agency needs models to express a) meaning and procedure of a logistics service it provides b) dependencies of the logistics service on a freight service that it uses and c) mappings of the service interactions to its internal business information systems. On the tactical level, service types have to be constantly maintained to keep track with organisational change *(service type adaptation).* Also on this level, partners have to be located for the types of service a participant is interested in as client *(service discovery)* or provider *(service publication).* On the operational level, partners have to be matched (by providers) and chosen (by clients) for service types *(service aggregation).* In some cases, providers additionally have to choose component service types matching the clients of composite services beforehand *(service composition).* During the actual service interaction procedure, terms and conditions of the service have to be ensured *(service coordination and control).* Additional flexibility can be reached by dynamic changes of service instances *(service instance adaptation).*

Generally for all levels, system software middleware is needed to arrange organisational environments of information- and communication technology (ICT) into a *cooperative information system* (Michelis et al., 1997) which realise services and provide support for the various tasks described above.

We refer to such a middleware as *service management system* and to the joint tasks of planning, building, and running of service oriented distributed applications as *service engineering.*

1.2 Current state of technology

Current techniques of service oriented computing are strongly focused on technology. While application-level (i.e. business) service support is out of their scope, they nevertheless pave the way towards it. The emerging Web Service standard (Tsalgatidou and Pilioura, 2002) provides interoperability between heterogeneous systems by leveraging the expressive power of XML to specify operational interfaces that can be accessed using open internet

communication. Thus, organisations can externalise their internal information systems as web enabled components. Those components provide interaction endpoints (subsequently called *ports*) to participate in automated inter-organisational cooperation. Concerning cooperation procedure, the service oriented model adopted by Web Services only defines a very basic type of interaction (i.e. 'broker triangle'). However, web service flow standards like BPEL4WS (Curbera et al., 2002) provide the means for individual definitions of basic interaction processes. This is the crossing point to more general research on cooperative, inter-organisational interaction-processes (e.g.(Baïna et al., 2003, Bussler, 2002, Schuster et al., 2000)) and workflow (e.g.(van der Aalst, 1999, Colombo et al., 2002, Chen and Hsu, 2000)), where several practical approaches for application-level services are located (e.g.(Mecella et al., 2001, Perrin et al., 2003, Casati et al., 2001)).

1.3 The Fresco Project

The FRESCO project is about foundational research on service composition (Piccinelli et al., 2003b). Its goal is to develop a framework of concepts and technologies that support organisations in playing the provider role for composite services. As a basis for composition, the focus is on the components first. Subsequently, a fundamental service model was developed that describes basic application-level services as classes of recurring cooperative interactions. The model was then implemented as a generic service engineering environment built on the Web Service family.

In the remaining parts of this paper, the Fresco approach will be detailed:

After the second part sketches a basic blueprint of our service engineering concepts, the third part introduces the Fresco Toolkit implementation. Finally, a summary and an outlook are given.

2. SERVICE ENGINEERING IN FRESCO

Fresco service engineering is based on a model that defines services as structured sets of cooperative interaction procedures. This model implies a specific architecture of service oriented applications that builds on an open, distributed component environment with service-enabling extensions. Subsequently, a service engineering environment provides a concise framework to plan, build, and run such service oriented distributed applications.

2.1 Service Model

The FRESCO Service model (Piccinelli et al., 2003a) defines a view on services that is provision-oriented and service-centric. Cooperation

procedures that constitute atomic, self-contained parts of a service-relationship are exposed by so called capabilities. In particular, capabilities represent purpose, interaction logic, and resulting artefacts of the cooperation between organisational roles. Thereby, capabilities define additional coordinative roles that introduce a level of indirection between participating roles. Unlike meta-level protocols, capabilities take the position of first-class participants (i.e. coordinators) that may be just virtually or effectively enforced. A service is made up by a set of such capabilities.

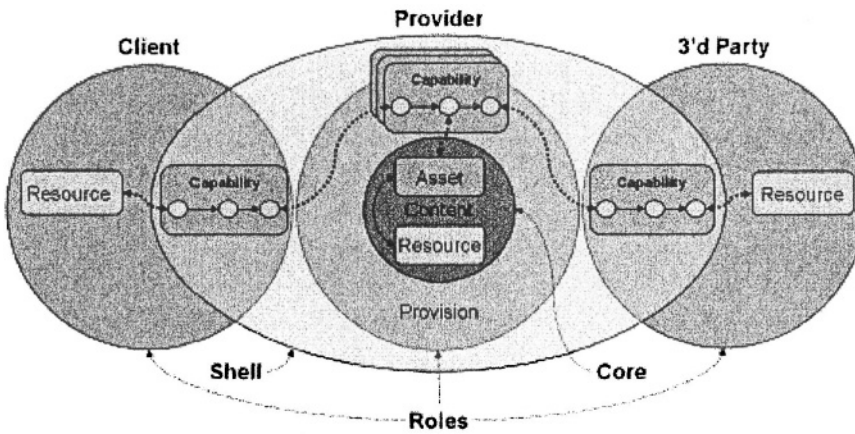

Figure 1. Fresco Service Model

An important feature of the model is a separation of capabilities in terms of service content and -provision. Content reflects the purpose of a service (e.g. moving goods). It is assumed that it arises from specific resources of the provider (e.g. internal processes, knowledge, people, machines, etc.). To represent service content, cooperation procedures, featuring interactions with such resources, are explicitly exposed as meaningful units of content (e.g. transport tracking...) by capabilities referred to as *assets*. Assets are degenerated in the sense that they don't represent cooperative interaction between roles but monologues of the provider (i.e. binding (Bussler, 2002)) that have to be provided to clients indirectly by other capabilities. Assets are grouped into a *service core* representing the complete content.

Provision addresses procedures that drive a service and make content available (e.g. negotiating terms and conditions, incorporating assets, etc.), whereby control is exclusively and proactive. Service provision capabilities (hence called "capabilities") are grouped around core assets in a layer called service shell. Within a shell, capabilities are mutually interrelated and share a common view on roles and provision-relevant information. Interrelations embody the overall behaviour of provision by defining the global interplay

of capabilities. A service is fully characterised by defining the basic core and, above all, the enabling shell (Figure 1). Our main focus is on the later. To realise this service model, associated technology has to focus on a) an architecture mapping the service notion to organisational ICT and b) an environment of mechanisms that facilitate service engineering tasks on top of it.

2.2 Service oriented architecture

For technology mapping we define a framework referred to as *service oriented architecture* (SOA). It provides a layer of abstraction that is assumed to wrap around diversified ICT systems in order to provide a homogeneous platform for service management. Service types are defined as *schemas* with respect to the SOA. Service instances can be run in any environment implementing the SOA framework.

In SOA, we assume that all organisational ICT resources of any role (e.g. client's ERP, provider's DBMS...), providing ports for service-related interactions, are represented by means of a homogeneous component model. Shell capabilities appear as glue between ports that reflects purpose, interaction logic and result. We represent this glue using workflow concepts based on the WfMC reference model (WfMC, 2002). Common patterns are prescribed to define capabilities as well as their structuring and interrelations by means of the workflow language XPDL, resulting in a *service schema.*

In particular, a capability maps to a set of workflow schemata describing a self contained unit of interaction. Ontology-associations define the *purpose* of *interaction logic* that emerges from the flow of interaction activities and results in data artefacts. Interaction activities can be defined for a participant (i.e. a role-associated component-port) to express cooperative procedure or for another capability workflow to express capability interrelation. Coherent sets of capability workflows are grouped together into packages with respect to a self contained task (e.g. negotiation capability, payment capability). The shell is given as a top-level package, where each capability is abstracted as a component type itself that realises the enclosed interaction flows and has a specific role assigned to it. Thus, various coordination concepts can be expressed including centralised- (orchestration) and distributed scenarios.

In brief, a schema specifies a partitioned set of highly interrelated components with precise interaction behaviour, where a subset A represents interacting participants and a subset B represents and enforces their interaction patterns. Service engineering is about planning, building and running B based on A.

2.3 Service engineering environment

Our concept of service engineering defines a set of basic engineering mechanisms that allow building customized extensions upon it. Besides *modelling,* the main problems addressed here are *adaptation, aggregation,* and *coordination.*

As services are inherently complex, we anticipate that support will be needed for their design, that is, a graphical *service modelling language* and tool, which help developers in creating service schemata. This is supposed to be the initial step of the service lifecycle, performed by the provider role.

Service schema management provides the functionality to process the schema programmatically. Beside storing and retrieving it, adaptation is its vital task. We adopt a rule based approach that provides a precise and systematic way to change schemata automatically. Back in the service lifecycle, the schema is subject of continuous static adaptation until eventually brought to action.

Then, it's the task of *service aggregation management* to create a service instance, based on the schema and a mapping of roles to actual participants. The main problem is to allocate resources of the participants according to the components associated to their roles, thereby optimising resource allocation while guaranteeing a constant and consistent flow of service procedures even when schema or participants change during provision. Initially, at least the provider is known and resources for an initial capability have to be allocated.

Service engines are components that manage the aggregation and coordination of capabilities they realise. The crucial problem is for participants to implement the capabilities of an engine while keeping the service context including associations to other engines and a homogeneous view on roles and data. We propose a generic implementation framework that can be parameterized with executable specifications generated from the schema. When all engines reach a final state the instance expires and the service lifecycle continues with a new round of static schema adaptation.

In addition to the core functions introduced so far, three other mechanisms are considered particularly useful: *service monitoring* to integrate the measurements of distributed sensors deployed throughout the service components into a coherent view of the overall service status, *security management* to allow controlling component access and delegating access privileges, and, finally, *type management* that defines a type system for generic software components and allows discovering compatibility and equivalence between them to support the handling of resources during service design and aggregation. Figure 2 gives an overview of all mechanisms and their respective relations.

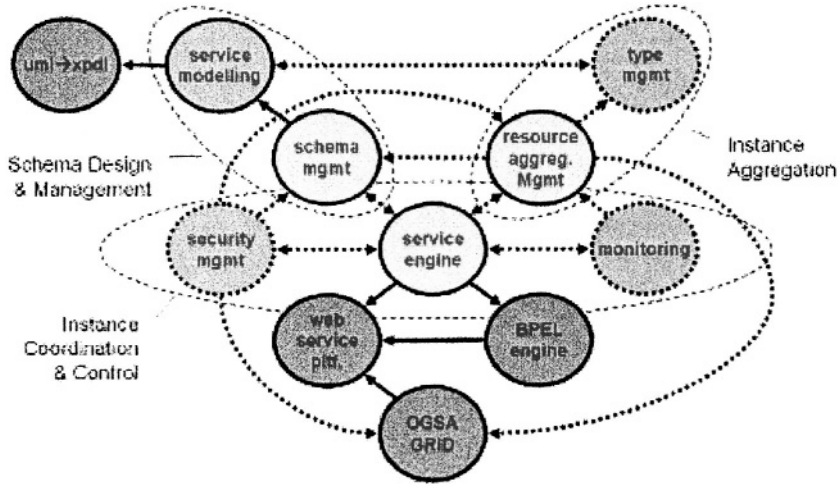

Figure 2. Service Engineering Environment

A vital characteristic of the overall engineering environment lies in the fact that all management mechanisms are first class components themselves. Thus changes can a) be made at provision time and b) arise from capabilities themselves. For example, a capability can lead to dynamic changes of participants (e.g. a new participant is introduced as a result of a brokerage capability) or dynamic schema adaptation (e.g. a payment procedure is changed as the result of a negotiation capability). Note, that this allows extending the service engineering mechanisms by realising them as capabilities.

3. THE FRESCO TOOLKIT

As a proof of concept, we developed the core functions of *service schema design, adaptation, aggregation,* and *coordination* in a prototype environment referred to as the *Fresco Toolkit (FrescoTK) 1.* This implementation is structured into parts related to *service schema* and *instance management.*

3.1 Service schema management

The focus of service schema management in Fresco is on the representation and organisation of interactive procedures that make up a service. A schema defines the complete shell of a service including roles and resources as well as the mapping of procedures to capabilities. More precisely, a service schema is realised as a structured transformable set of abstract workflow definitions. We use XPDL, where generic workflow elements are defined in the context of packages that can again refer to other

packages thus allowing the definition of coherent structures. A service schema contains a) a root package representing the service shell and declaratively defining the service capabilities, b) one set of packages for each capability that defines its interactive procedures, and c) one context package that defines a common context of roles, data structures, and resources. The FrescoTK *Schema Manager* component (Figure 3) holds generic specifications of various service schemata and makes them programmatically accessible. Its vital characteristic is the ability to apply a variety of transformations to them that allow for controlled changes of service structures as well as for the logic of interactive procedures. Moreover, it is possible to change the representation of procedures into executable format.

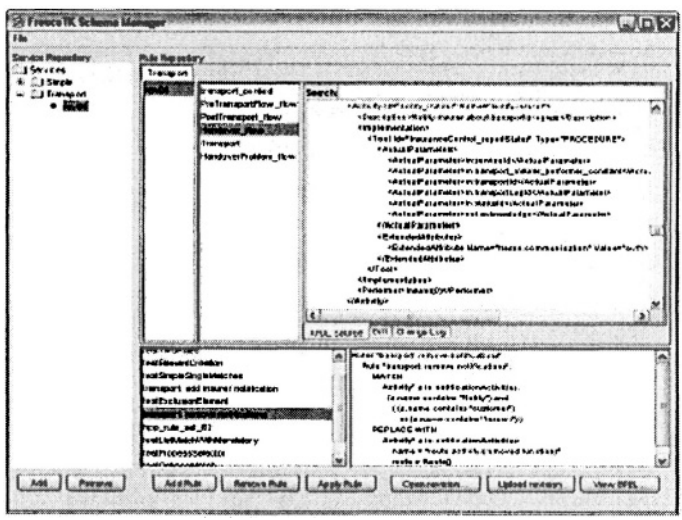

Figure 3. FrescoTK service schema tool

Evolutionary adaptation is supported by means of a language for change rules that are enforced by a rule engine within the schema manager. It allows *matching* arbitrary patterns in XPDL workflow process descriptions and *removing* or *replacing* those matches with newly created process elements into self contained revisions. However, the procedural logic of capability components, given in XPDL, has to be transformed into a format that can be executed by an engine. Those engines are used as active components that enforce provision procedures at runtime (see 3.2). The transformation is based on the fact that most workflow languages share a set of core concepts with common semantics (see (van der Aalst, 2003)). We chose the emerging BPEL4WS standard as our execution format and defined a mapping to it from XPDL (a full coverage can be found in the FrescoTK documentation).

3.2 Service instance management

Service instance management comprises organisation of participants and resources for service instances. The FrescoTK *Aggregator component* evaluates service schemata for involved roles and necessary resources (Figure 4).

During service execution, all roles have to be assigned to participants and each of them has to provide the resources associated with its roles. An individual strategy can be chosen for each service that specifies how to do role assignment and resource creation in terms of schedule and execution model.

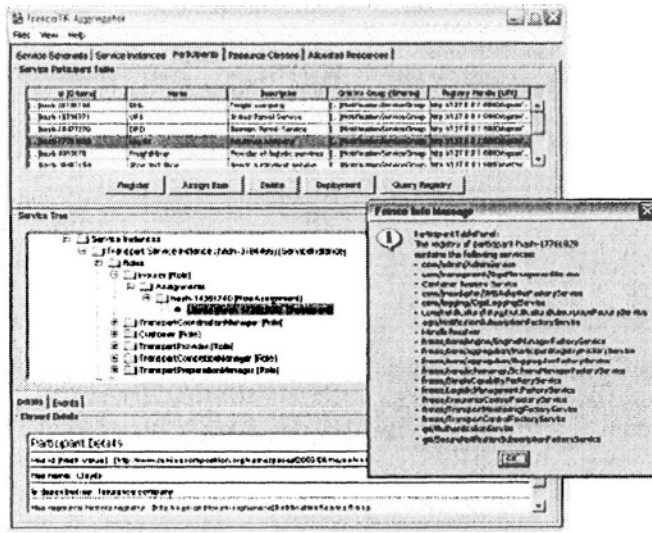

Figure 4. FrescoTK Aggregator

In FrescoTK the SOA is based on OGSA (Foster et al., 2002) as the component model. Thus, service related resources as well as schema management, aggregation, and engine components are built as GRID services. Engine components are realised by a BPEL engine that executes process specifications generated by the schema manager (Figure 5). The engine is wrapped as a grid component by adapters and proxies that are automatically generated for each capability. They bridge the gap between stateless Web Services and long lived Grid Services using the aggregator to resolve references of individual resources.

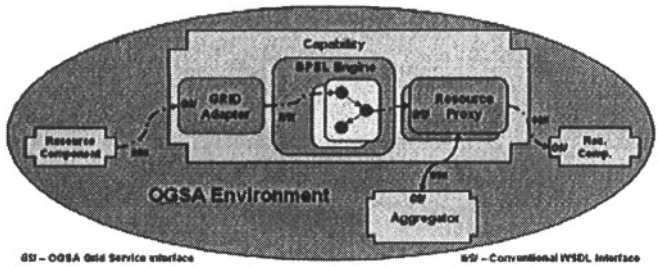

***Figure* 5. Capability Grid Architecture**

4. CONCLUSION

As inter-organisational relationships and cooperation increase in advanced e-business (and other similar) applications both in terms of quantity and of quality, the need for new classes of distributed applications arises that allow their effective and efficient management. In this paper, we focus on recurring cooperation scenarios between changing autonomous participants and system support for them based on service-oriented distributed applications.

While, in such a context, a suitable technological foundation is already in place to interconnect the participants, adequate support for, e.g., planning, building, and running such solutions is still missing. Therefore, we propose a service model based on advanced Web service and Grid Service technology and address a set of problems realising it within a basic service engineering approach. This approach applies process theory and workflow concepts to specify, aggregate, enact, and adapt services as interaction patterns between distributed resources. In particular, we adopt a homogeneous view on resources, coordination-, and engineering mechanisms that allows for a degree of introspection and dynamic self-adaptation.

We claim that this concept is powerful enough to implement complex service scenarios with customized requirements. In future work, we will use the service engineering mechanisms to examine models and mechanisms for service composition that allow relating and connecting the capabilities of composite services to the capabilities of their service components.

REFERENCES

Baïna, K., Tata, S. and Benali, K. (2003) A Model for Process Service Interaction, In Busines Process Management International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003. Proceedings(Ed, Weske, M.) Springer, pp. 261 ff.

Bussler, C. (2002) Behavior abstraction in semantic B2B integration, In Conceptual Modeling for New Information Systems Technologies. ER 2001 Workshops. HUM ACS, DASWIS, ECOMO, and DAMA. Revised Papers Lecture Notes in Computer Science Vol.2465. 2002(Ed, Hunt, I.) Springer Verlag, Berlin, Germany, pp. 377-89.

Casati, F., Sayal, M. and Ming Chien Shan (2001) *Developing e-services for composing eservices,* In *Advanced Information Systems Engineering. 13th International Conference, CAiSE 2001. Proceedings Lecture Notes in Computer Science Vol.2068. 2001*(Ed, Norrie, M. C.) Springer Verlag, Berlin, Germany, pp. 171-86.

Chen, Q. and Hsu, M. (2000) *Inter-Enterprise Collaborative Business Process Management, HPL-2000-107, Software Technology Laboratory, HP Laboratories Palo Alto*

Colombo, E., Francalanci, C. and Pernici, B. (2002) *Modeling Coordination and Control in Cross-Organizational Workflows,* In *Proc. CoopIS/DOA/ODBASE 2002*(Eds, Meersmann, R. and Tari, Z.) Springer, pp. 91 ff.

Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S. and Weerawarana, S. (2002) *Business Process Execution Language for Web Services, V 1.0, BEA, IBM, Microsoft*

Foster, I., Kesselman, C., Nick, J. and Tuecke, S. (2002) *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG, Global Grid Forum* ISO/IEC-JTC1/SC21 (1995) *Basic Reference Model of Open Distributed Processing -- Part3: Architecture. International Standard, 10746-3, ISO*

Mecella, M., Pernici, B., Rossi, M. and Testi, A. (2001) *A Repository of Workflow Components for Cooperative e-Applications,* In *Proceedings of the 1st IFIP TC8 Working Conference on E-Commerce/E-Business (Salzburg, Austria, 2001)*BICE Press, pp. 73-92.

Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A. H. H. and Elmagarmid, A. K. (2003) *Business-to-business interactions: issues and enabling technologies,* The VLDB Journal, (Springer, Online First, April 3, 2003).

Michelis, G. D., Dubois, E., Jarke, M., Matthes, F., Mylopoulos, J., Papazoglou, M. P., Pohl, K., Schmidt, J., Woo, C. and Yu, E. (1997) *Cooperative Information Systems: A Manifesto,* In *Cooperative Information Systems* (Ed, Papazoglou, M., Schlageter, G.) Academic Press.

Perrin, O., Wynen, F., Bitcheva, J. and Godart, C. (2003) *A Model to Support Collaborative Work in Virtual Enterprises,* In *Business Process Management International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003. Proceedings*(Eds, Aalst, W. M. P. v. d., Hofstede, A. H. M. t. and Weske, M.) Springer, pp. p. 104 ff.

Piccinelli, G., Zirpins, C. and Gryce, C. (2003a) *A Provision-Centric Model for Electronic Services,* In *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-2003)*IEEE Computer Society, pp. 113-116.

Piccinelli, G., Zirpins, C. and Lamersdorf, W. (2003b) *The FRESCO Framework: An Overview,* In *2003 Symposium on Applications and the Internet Workshops (SAINT 2003 Workshops)* IEEE Computer Society, pp. 120-123,

Schuster, H., Georgakopoulos, D., Cichocki, A. and Baker, D. (2000) *Modeling and Composing Service-Based and Reference Process-Based Multi-enterprise Processes,* In *Proc CAiSE 2000*(Ed, Bergman, L.) Springer, pp. 247-263.

Tsalgatidou, A. and Pilioura, T. (2002) *An overview of standards and related technology in Web Services,* Distributed and Parallel Databases, 12, 135-62.

van der Aalst, W. M. P. (1999) *Process-oriented architectures for electronic commerce and interorganizational workflow,* Information Systems, 24, 639-71.

van der Aalst, W. M. P. (2003) *Don't go with the flow: Web services composition standards exposed,* IEEE Intelligent Systems, 18.

WfMC, (2002) *Workflow Management Coalition,* http://www.wfmc.org, 1.5.2003