A Coordination Space Architecture for Service Collaboration and Cooperation

Claus Pahl, Veronica Gacitua-Decar, MingXue Wang, and Kosala Yapa Bandara

Lero - The Irish Software Engineering Research Centre
School of Computing, Dublin City University
Dublin, Ireland
[cpahl|vgacitua|mwang|kyapa]@computing.dcu.ie

Abstract. With software services becoming a strategic capability for the software sector, service engineering needs to address integration problems based on support that helps services to collaborate and coordinate their activities. The increasing need to address dynamic and automated changes - caused by on-demand environments and changing requirements - shall be answered through a service coordination architecture based on event-based collaboration. The solution is based on a service coordination space architecture that acts as a passive infrastructure for event-based collaboration. We discuss the information architecture and the coordination principles of such a collaboration environment.

1 Introduction

Service-oriented architecture (SOA) as a methodological framework, using Web services as the platform technology, supports a range of integration problems. SOA aims at providing a service-based infrastructure for interoperable systems development and integration. However, trends such as on-demand and service outsourcing, which are supported by Web services as the platform technology, pose further challenges for software and systems engineering.

We introduce a platform for service collaboration that makes a significant step from static service architectures (based on Web service orchestration and choreography) to dynamic coordination and collaboration in open service communities [1]. Particularly scalability and dynamic flexibility are limitations of current orchestration techniques. WS-BPEL orchestrations require a static coupling of providers and users of services. Even if a mediator would manage WS-BPEL processes dynamically, this could create a bottleneck. The coordination solution based on a coordination space addresses the need to support dynamic collaborations. This solution enables the self-organisation of large service communities, thus addressing the scalability problem. In addition, the proposed platform can enable flexible dynamic composition of service architectures.

In contrast to existing registry and mediation solutions, where providers initially publish their services and where clients search for suitable services, here the approach is reversed - changing from a pull-mode for the client to a push-mode where the client posts requests (into the coordination space) that can be taken on by providers. We discuss the information models, the coordination principles and the architecture of the coordination space approach.

Different coordination models has been proposed, e.g. based on the Linda tuple space model [2, 3, 4]. More recently, domain- and application context-specific solutions [5, 6, 7] and approaches based on semantic extensions are investigated [8, 9]. The latter ones have also been applied to service composition and mediation. We built up on these semantic mediation approaches by adding a process component and by linking this to a tuple coordination technique.

The next section discusses service collaboration using motivating scenarios and analysing current technologies. In Section 3, we describe technical aspects of the coordination space in terms of information models, coordination principles and the architectural issues. Section 4 discusses implementation and evaluation. We discuss related work in Section 5 before ending with some conclusions.

2 Service Collaboration

Our aim is to enable an integrated collaboration of service communities. These service-based applications often exhibit a dynamic nature of interaction, which requires novel techniques for identification of needs and behaviours and the adaptation of provided services to requested needs. The coordination of activities between communities of users and providers needs to be supported.

2.1 Motivating Scenarios

Two scenarios shall motivate our solution. Firstly, localisation is a multi-lingual Internet application scenario that involves distinct users and providers joined together through a common workflow process. With the emergence of multi-lingual knowledge exchange and social networking sites, localisation becomes an every-day activity that needs to be supported by coordination infrastructure.

– Sample objects are text documents (software manuals or legal documents) to be translated or translation memories to be used in the translation process.
– Sample activities that process the objects include translation (several different techniques may be supported), the creation of translation memory (e.g. crowd-source) as well as translation evaluation and correction.
– The activities can be combined into different localisation workflow processes.

For example, a software developer requires the translation of a software manual for a new market. The developer makes the document to be translated (object) together with a goal (the required language and quality of translation) available through the coordination space. Translators, represented by a provider service can accept available translation tasks, might even in turn request specific translation memories (e.g. through a direct retrieval request or crowd-sourced, both via the coordination space). After finishing the translation the translator makes the translated document available for further processing with an evaluation goal to validate the achieved quality. Another service provider takes on this task. While this process is ongoing, the infrastructure can incrementally identify the sequence of activities as being part of a standard localisation workflow pattern and can afterwards more closely guide and govern the process. At any stage, the infrastructure can also aid by mediating between service users and providers, e.g. converting between different data formats used.

Secondly, customer care is a classical Enterprise software scenario (layered on top of a full software system) that can be enhanced through distributed, on-demand collaboration infrastructure.

– Sample objects are software objects, problem descriptions and help files.
– Activities include explanations or activities to repair or adapt software.
– Two sample processes are a software help wizard that guides an end-user through a number of steps to rectify a problem and a customer care workflow that in a number of steps identifies a problem, decides on a resultion strategy and implements the latter.

Initially, a software user asks for help by posting a help request referring to the software component and the problem in question. An analysis service takes on the first task and determines whether explanation and guidance is sufficient or whether the software itself needs to be adapted. In both cases, new requests (objects and goals) are generated. In the first case, the discovery of suitable responses (e.g. by retrieving and/or assembling help files) is the aim. In the second case, software changes need to be implemented. Again, identifying the process pattern allows more targeted processing of the initial goal.

2.2 Discussion of Existing Technology

The current approach to service collaboration is to develop service processes that are orchestrations or choreographies of individual services. Service orchestrations are executable process specifications based on the invocation of individual services. WS-BPEL, the business process execution language, is the most prominent example, which has become a de-facto standard. A WS-BPEL process is executed through an execution engine. Choreographies, e.g. defined in terms of the choreography description language WS-CDL, are different from orchestration in that they describe assemblies of services as interactions between different partners from a global perspective. It is however similar in that predefined static process compositions are defined. While this has been successful for intra-organisational software integration, limitations can be observed:

– Inflexible nature: Common to both is the static nature of these assemblies, requiring them to be pre-defined and which can only be alleviated to some extent through dynamic adapter generation [10]. This makes the current dynamic nature of service architecture difficult to support. Orchestrations are based on a number of pre-defined partners, not providing sufficient flexibility for future Internet requirements. In traditional SOA, services are orchestrated centrally,

assuming that what should be triggered is defined in a predefined business process - which does not account for runtime events that occur elsewhere.
– Lack of scalability: Orchestrations and choreographies are simple process programs without abstraction mechanisms, thus restricting the possibility to define complex and manageable system specifications. Adding procedural abstraction mechanisms as in other programming languages would not suffice to deal with the aforementioned dynamic and flexibility problems.

A look at the two scenarios illustrates the identified limitations. Increasing flexibility of composition by allowing partners to dynamically join or leave the provider community is not possible using the classical approach. The localised requesting and providing of services (by asking for activities to be executed on objects to achieve a goal) avoids complex, pre-defined process definitions, thus making the coordination more scalable through self-organisation.

While Web service platform and service computing technologies exist, approaches that are suitable to support collaborative, dynamic applications are lacking and platform infrastructures are only beginning to mature. Particularly, scalability and dynamic flexibility, i.e. suitability for open and collaborative infrastructures and applications, are limited due to the restrictive nature of current service composition, collaboration and interaction techniques such as orchestration and choreography languages like WS-BPEL [11].

3 Service Coordination Space Architecture

The core concept of our solution to address flexibility and scalability of service collaboration is a coordination space. This coordination space acts as a passive infrastructure to allow communities of users and providers to collaborate through the coordination of requests and provided services, see Fig. 1. It is governed by coordination principles, which are event-driven at the core:

– tasks to perform an activity on an object occur in states
– services collaborate and coordinate their activities to execute these tasks
– advanced states are reached if the execution is permitted and successful

The central concepts of the coordination space solution are objects, goals (reflecting the outcomes of activies) and processes that are seen as goal-oriented assemblies of activities. Service requesters enter a typed object together with a goal that defines the processing request. Service and process providers can then select this processing task. The coordination space is a repository for requests that allows interaction between requesters and potential providers to be coordinated. The knowledge space provides platform functionality, e.g. the event coordination and request matching, following the platform-as-a-service idea. The Web service platform provides with UDDI a static mediator component that is not suitable to support dynamic collaboration. Our proposal is also different from UDDI in a more fundamental way. It changes the perspective of the client from a pull- to a push-approach. Instead of querying (pull) the repository for

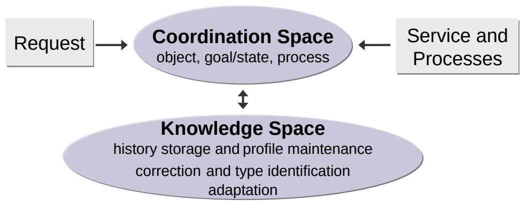
Request
Coordination Space
object, goal/state, process
Service and Processes
Knowledge Space
history storage and profile maintenance
correction and type identification
adaptation

Fig. 1. Coordination and Knowledge Space Architecture.

suitable entries that providers might have published before, the submit (push) requests here, which in turn are picked up by providers.

3.1 Information Architecture

Users are concerned with the processing of objects. In classical enterprise scenarios these objects are electronic documents passing through business processes, but within future Internet applications, the object notion will broaden, capturing any dynamic, evolving entity that is part of a process as an ongoing activity. The central concepts of the information architecture are objects, goals and processes:

– Objects play a central role. Changing, evolving objects are dynamic entities that represent an end-to-end view. This follows trends to focus on structured objects and documents as the central entities of processing, as proposed by ebXML and other business standards.
– Goals are declaratively specified, expressing the requested result of processed objects [12]. Essentially, the aim is to allow users and providers to refer to them declaratively, e.g. in the form of a goal-oriented user request (requesting object processing) and to enable semantic goal-based matching.
– The process notion refers to business and workflow processes. States of the process are points of variation for objects: data evolves as it passes through a process. Goals relating to objects are expressed in terms of states of the processes where a process state is considered at the level of individual object modifications. The link to objects is provided via states of processes. Process-centricity is still the central feature of service coordination, thus we retain the compositional principle of Web service technology. Common and successful patterns can be captured and reused [13].

For instance, service computing as an information processing and management infrastructure is becoming of significant importance that would benefit from requests being formulated in terms of the underlying information objects being processed, an abstract specification of goals and the process that the individual processing activities are embedded in.

We propose a semantically enriched information architecture here capturing object structure, object modification states and an object evolution process [14, 15]. Ontologies with descriptive and operational layers through an encoding of dynamic logic in a description logic will provide the foundations for the object and process specification framework [16]. This allows us to include behavioural and temporal aspects into the core, static ontology framework capturing objects.

– Objects types are expected to be represented as concepts in a domain ontology. Objects in the form of XML data schemas represent the object type. A composition relationship becomes a core generic relationship for our request ontology (in addition to the traditional subsumption-based taxonomic relationship). Structural nesting of XML elements is converted into ontological composition relationships.
– Goals are properties of the object concepts stemming from the ontology (covering domain-specific properties as well as software qualities). Goals are expressed in terms of ontology-based properties of objects (concepts), denoting states of an object reached through modification (processing).
– Processes are based on a service process ontology with specific service process composition operators (like sequencing ';', parallel composition '||', choice '+' or iteration '!' – see [16] for details) as part of the ontology. Processes are specified based on input and output states linked to goals as properties.

This shall be illustrated using the second scenario used earlier on. A typical software component as an object in the context of the customer care and maintance scenario has properties such as deployed, analysed, or redeveloped. A maintenance process could be expressed as a cyclic process *!(deployed; analysed; redeveloped)*, which defines an iteration of the 3-sequence of activities.

3.2 Coordination Principles

For goal-driven, event-based collaboration of services, coordination space functionality and event handling are important aspects. The functionality provided by the coordination space essentially follows established coordination approaches like tuple spaces [2, 3, 4] by providing deposit and retrieval function for the space elements - tuples which consist of object type, goal and supporting process. Specifically, one deposit and two retrieval functions are provided:

– deposit(object!, goal!, process!), where the parameters are defined as above in Section 3.1, is used by the client and deposits a copy of the tuple [object, goal, process] into the coordination space. The exclamation mark '!' indicates that values are deposited. The process element is optional; it can be determined later to guide individual processing activities. deposit returns the number of equal tuples already deposited (the tuple will be deposited nonetheless and internally a sequence number and an ID identifying the depositor is kept).
– meet(object?, goal?), with parameters as above, is used by a service provider and identifies a matching tuple in the coordination space. The question mark indicates that abstract patterns are provided that need to be matched by concrete deposited tuples. Ontological subsumption reasoning along the object concept hierarchy enhances the flexibility of retrieval by allowing subconcepts to be considered as suitable matches. A subconcept is defined as a subclass of the concept in terms of the domain ontology, but it also takes the composition

properties (see explanation of structural composition in the previous section) into account, i.e. requires structural equality [16]. Operation meet does not block or remove the tuple; it does not have any side-effect. meet returns a boolean value indicating whether the match has been successful.

– fetch(object?, goal?), with parameters as above, is used by a service provider to identify a matching tuple. Ontological subsumption reasoning is used as above. fetch does remove the tuple, blocking any further access to the tuple. fetch also returns the number of matching tuples. The coordination space will select the tuple that is deemed to be the closest subsumption-based match, i.e. conceptually the closed on the ontological hierarchy.

meet is used to inspect the tuple space; fetch is used if a requested task is taken on and shall be interpreted as a commitment to achieve the specified goal.

The event handling complements the coordination space manipulation through the three functions: this includes registration mechanisms for requestors (in order to be allowed to deposit object processing requests) and service providers (in order to allow to discover, match and retrieve processing requests). The event handling provided by the infrastructure allows providers to register profiles, according to which notifications of matching request tuples are sent out.

In a concrete situation, a service requestor would deposit a request tuple, service providers able to fulfil the request (determined through meet) would compete to take on the task. The successful provider would block the request (using fetch) for others. He could, in turn, use the coordination space to request further internal tasks to be satisfied, as illustrated by the motivating scenarios. A sample scenario is illustrated in Fig. 2.

These coordination principles can be formalised and implemented using Event-Condition-Action (ECA) rules. These ECA rules govern the execution of the coordination space. For instance, with *E = deposit(...), C = meet(...), A = fetch(...)*, the reaction of a service provider (blocking retrieval of a matching tuple deposited by a requester) can be defined. In this simple form, the rules expresses synchronisation between requestor and provider and semantic matching. However, the rules can also be used to deal with SLAs, governance and compliance and competition aspects if additional parameters are taken into account during matching. In general, the rules allow constrained and controlled self-management of coordination.

In Fig. 2, a process is emerging from the sequence of events, indicated through the numbered coordination space operations. The processes can be automatically identified and used to guide and constrain further processing. The schematic example follows the second scenario (customer care) and abstracts its activities:

1. client deposits the help request (problem description object with guidance as the goal)
2. one service provider meets and fetches the request
3. provider creates and deposits two more requests - one to create an explanation of the problem, the other to determine whether the software needs to be modified (software entity as object and analysis request as goal)
4. these tuples are in turn fetched by other providers

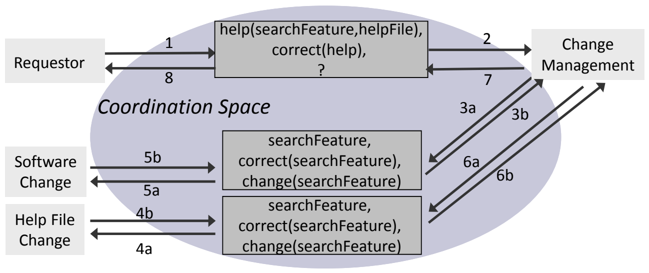
Requestor
1
8
help(searchFeature,helpFile), correct(help), ?
2
7
Change Management
Coordination Space
3a
3b
6a
6b
Software Change
5b
5a
searchFeature, correct(searchFeature), change(searchFeature)
Help File Change
4b
4a
searchFeature, correct(searchFeature), change(searchFeature)

Fig. 2. Coordination Example.

5. these providers then deposit solutions
6. which are fetched by the initial provider
7. who in turn deposits an overall solution
8. which is finally used by the client.

The first provider might recognise, supported by the knowledge space, that a common process pattern can be applied [13] (we use the notion of a type to technically capture these patterns). The patterns can provide compliance with process regulations and guarantees of quality. A repository of common processes held in the knowledge space can be used to identify these and use the process to fill the optional process component of the tuple. In this case, the process will indicate a sequence of subactivities (some composed in parallel such as steps 3a and 3b), linked to subgoals, which decompose the original higher-level comprehensive rule - steps 3 to 6 in the example.

3.3 Architecture and Implementation

The functionality separation into basic coordination and advanced semantic analysis support is implemented by a joint architecture of two components - the coordination space and the knowledge space - which support the core coordination and analysis activities, respectively. This architecture is built on the software-as-a-service (SaaS) principle. The structure and functionality of these two spaces shall be described here.

– Coordination Space: The tuples entered consist of the object type, the goal and the process type - the latter identifies the process pattern and may not be available initially and could be determined by the knowledge component. An event model based on the ECA rules governs the operation of the coordination space. The event coordination model comprises rules on events such as deposit goal, identify type, accept task, etc. The functions of the coordination space include the storage of artefacts, the generation and handling of events, and the monitoring activities and collection of relevant information for storage.
– Knowledge Space: This repository for process types and ontologies defines the semantic structure for data objects and process behaviour. The functions of

the knowledge space (provided as services) include: the matching support and the process type determination. In the localisation scenario, it can provide support to identify the workflow process to govern the overall execution.

The functionality of the coordination and knowledge spaces is implemented in the form of infrastructure services. The Coordination Engine implements the event model-based coordination functions that allow requests to be posted and tasks to be assigned. The Knowledge Services implement monitoring, collection and storage as core functions and the high-level analysis features. These services are based on Java implementations exposed as services. The coordination engine provides two APIs - a deposit API for requestors and a retrieval API for providers, see Fig. 2. The interaction with engines is based on classical service platform technology, i.e. services specified in WSDL and interaction through the SOAP protocol using the ActiveBPEL engine as the core platform component.

The coordination space idea is the central contributor to increased flexibility, as the coordination space, compared to static orchestrations, enables flexibility through de-coupling requester and provider and the possibility to compose services dynamically. The separate knowledge space provides scalability in terms of the infrastructure support as these services can be provided independent of core coordination functions. It follows what has made middleware successful.

4 Discussion and Evaluation

Part of the implementation and deployment aspects is the consideration of hosting scenarios. Beyond the technical implementation of the spaces as services, the question who hosts these needs to be answered as these are meant to act as intermediaries between requesters and providers. However, proposals exist for instance for hosting UDDI repositories as a static and more manually operated form of an intermediary or, more recently, for semantically enhanced mediators for semantic service matching and semantic spaces that demonstrate the viability of the hosting scenario. Once this infrastructure, involving a third part, is in place, value-added functions to coordination can easily be added to the knowledge space in the software-as-a-service spirit.

The two scenarios used to motivate the usefulness of the proposed approach have also been used by us to validate the conceptual architecture of coordination spaces, specifically demonstrating the benefits of the coordination principles and protocol aspects. Both scenarios are based on real-world applications. Traditionally implemented solutions for both exist as demonstrator applications within a research centre we participate in. The benefit of this evaluation setting in order to demonstrate the benefits of a novel platform technology is twofold:

- The demonstrator scenarios have been developed in close collaboration with industrial partners and represent state-of-the-art implementations.
- The existence of the technologically advanced solutions allows a comparison between the existing project demonstrators and their reimplementation based on the proposed infrastructure using the prototype discussed earlier.

While full applications have not been developed yet, we have focussed our evaluation on a systematic performance analysis of the scenarios with a larger number of clients and providers (> 20 for each), which allows us to conclude significant scalability benefits using the proposed collaboration approach as indicated in the discussions earlier. We have observed an acceptable performance overhead of around 10 % for coordination activities compared a hardwired WS-BPEL composition. In a traditional solution, either a significant number of WS-BPEL processes would have to be provided, or a mediator that explicitly executes provider selection would have to be added (which might create a bottleneck for larger service communities in contrast to the passive coordination space where some selection actitivies is carried out by providers).

Another key result is that decoupling through the coordination space provides more flexibility. The push-model allows for more dynamic composition as the example in Fig. 2 illustrates. The coordination primitives allow a localisation of collaboration activities, thus enhancing the scalability through local self-management. The semantic support infrastructure enhances flexibility through the matching techniques and allows for higher degrees of automation.

5 Related Work

The coordination paradigm applied here is a fundamental change to existing service discovery and matching approaches. Coordination models have been widely used to organise collaboration. The Linda tuple space coordination model [2] has influenced a wide range of variations including our on work on concurrent task coordination in programming languages [3], which in turn has influenced the solution here. More recently, domain- and application context-specific solutions and approaches based on semantic extensions have been investigated. However, the specifics of distributed on-demand environments has not yet been addressed. In particular over the past years, coordination has received much attention [5, 6, 7] due to the emergence of collaborative ICT-supported environments, ranging from workflow and collaborative work to technical platforms such as service collaboration. The latter ones have also been applied to service composition and mediation. Only recently, event handling has been considered in the SOA context in the form of event-driven SOA. In [17], an ontology-based collaboration approach is described that is similar to ours in that it advocates a push-style of coordination. We have added to semantic mediation approaches like [8, 17, 18] by including a process notion as a central components of request tuples [14], supported by a process-centric ontology languages. Through the goal/state link, this process context is linked to the request coordination technique focussing on objects are primary entities.

An important contributor to the overall success is backward compatibility. We can realise BPEL-style interaction as a collaboration in our coordination spaces. A request tuple can be automatically generated from a BPEL process by using the BPEL input parameters as the object and the BPEL process abstraction as the process element as the process type of the tuple. The goal would in this case be left unspecified, and provider matching would be individually carried out for each invocation in the BPEL process.

6 Conclusions

Integration and coordination of services is at the core of recent dynamic service architectures. However, their structural, inherent inflexibility makes changes and evolution difficult. Current service computing platforms suffer from flexibility and scalability problems, which can be overcome through the proposed coordination space. This coordination technologically significantly enhances the collaboration and also competition capabilities of the Web service platform, particularly in the context of dynamic applications.

The proposed coordination technology aims directly at the core challenges of service architecture, such as the lack of a flexible mechanism for service collaborations and the need for variability and flexibility in dynamic settings. Decoupling achieves flexibility. Scalability is achieved through a passive coordination architecture with reduced coordination support - which, however, necessitates the cooperation of providers to engage and pro-actively use the coordination space as a market place. The aims are to achieve cost reductions for service interoperation covering design and deployment; operation, maintenance and evolution, while still enabling flexibility for collaborating service architectures.

While we have defined the core coordination principles here, the range of supporting features through the knowledge space needs to be investigated further. Part of this are fault-tolerance features supporting self-management and semantic techniques deducing object and process types from possibly incomplete information [19]. Trust is a related aspects that needs to be addressed. We have occasionally indicated advanced functionality of the knowledge space, e.g. the automated identification of processes based on stored process history or the possibility to consider non-functional aspects reflected in profiles and context models during matching. This requires further investigation.

Acknowledgment

This work was supported by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie) and grant 07/RFP/CMSF429 CASCAR.

References

1. J. Rao and X. Su. A Survey of Automated Web Service Composition Methods. Intl. Workshop on Semantic Web Services and Web Process Composition 2004. Springer LNCS 3387, Pages 43-54, 2005.

2. D. Gelernter. Generative Communication in Linda. ACM Transactions on Programming Languages and Systems 7(1):80-112. 1985.
3. E.-E. Doberkat, W. Hasselbring and C. Pahl. Investigating Strategies for Cooperative Planning of Independent Agents through Prototype Evaluation. In Proc. Intl Conf on Coordination Models and Languages. Springer, LNCS 1061, 1996.
4. E.-E. Doberkat, W. Franke, U. Gutenbeil, W. Hasselbring, U. Lammers and C. Pahl. PROSET - Prototyping with Sets, Language Definition. Software-Engineering Memo 15, Universitt GH Essen, 1992.
5. B. Johanson and A. Fox. Extending Tuplespaces for Coordination in Interactive Workspaces. Journal of Systems and Software 69(3), 243-266. 2004.
6. Z. Li and M. Parashar. Comet: A Scalable Coordination Space for Decentralized Distributed Environments. In Proc. Intl. Workshop on Hot Topics in Peer-To-Peer Systems HOT-P2P. IEEE, 104-112. 2005.
7. D. Balzarotti, P. Costa, G.P. Picco. The LighTS tuple space framework and its customization for context-aware applications. Web Intelligence and Agent Systems 5(2): 215-231. 2007
8. L. Nixon, O. Antonechko and R. Tolksdorf. Towards Semantic tuplespace computing: the Semantic web spaces system. In Proceedings of the 2007 ACM Symposium on Applied Computing SAC '07. ACM, 360-365. 2007.
9. C. Pahl, Y. Zhu. A Semantical Framework for the Orchestration and Choreography of Web Services. Proceedings of the International Workshop on Web Languages and Formal Methods (WLFM 2005). Electronic Notes in Theoretical Computer Science 151(2):3-18. 2006.
10. A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. Proc. ICSOC06. Pages 27-39. Springer LNCS 4294. 2006.
11. C. Pahl. A Conceptual Architecture for Semantic Web Services Development and Deployment. International Journal of Web and Grid Services 1(3/4):287-304. 2005.
12. B. Andersson, I. Bider, P. Johannesson, and E. Perjons. Towards a formal definition of goal-oriented business process patterns. BPM Journal 11:650-662. 2005.
13. V. Gacitua-Decar and C. Pahl. Automatic Business Process Pattern Matching for Enterprise Services Design. 4th International Workshop on Service- and Process-Oriented Software Engineering (SOPOSE-09). IEEE Press. 2009.
14. C. Pahl. A Formal Composition and Interaction Model for a Web Component Platform. ICALP'2002 Workshop on Formal Methods and Component Interaction. Malaga, Spain. Elsevier Electronic Notes on Computer Science ENTCS, Vol. 66, No. 4. July 2002.
15. C. Pahl. A Pi-Calculus based Framework for the Composition and Replacement of Components. Proc. Conference on Object-Oriented Programming, Systems, Languages, and Applications OOPSLA'2001 - Workshop on Specification and Verification of Component-Based Systems. Tampa Bay, Florida, USA. ACM Press. 2001.
16. C. Pahl, S. Giesecke and W. Hasselbring. Ontology-based Modelling of Architectural Styles. Information and Software Technology. 1(12): 1739-1749. 2009.
17. W.T. Tsai, B. Xiao, Y. Chen and R.A. Paul. Consumer-Centric Service-Oriented Architecture: A New Approach. In Proc. Workshop on Software Technologies for Future Embedded and Ubiquitous Systems. Pages 175-180. 2006.
18. C. Pahl. Layered Ontological Modelling for Web Service-oriented Model-Driven Architecture. European Conference on Model-Driven Architecture - Foundations and Applications ECMDA'2005. LNCS 3748. Pages 88-102, 2005.
19. M. Wang, K. Yapa Bandara and C. Pahl. Integrated Constraint Violation Handling for Dynamic Service Composition. IEEE International Conference on Services Computing SCC 2009. IEEE. 2009.