Service Typing in Collaborative Systems

Toni Ruokolainen, Lea Kutvonen

Dept. of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 UNIVERSITY OF HELSINKI
FINLAND
toni.ruokolainen@cs.helsinki.fi,lea.kutvonen@cs.helsinki.fi

Abstract. Advanced enterprise computing environments are computing systems comprised of networked business services and interwork-enabling infrastructure services. Modern networked enterprises require flexibility and openness of computing systems to tolerate changes in ever-changing technology and business domains, and to gain a competitive edge. Such requirements, however, make it difficult to achieve interoperability between distinct enterprise-information systems. This paper discusses the use of service typing as the conceptual basis for achieving interoperability in enterprise computing environments. Type management infrastructure services are considered as an essential part of a future B2B-middleware platform. First the notions of service typing and type management infrastructure are introduced. After this brief introduction, the use of service typing is motivated. After this more detailed descriptions of service types and type management functionality are given. We conclude with a brief discussion and comparison to related fields of research.

1 Introduction

Advanced computing environments are inter-enterprise computing systems consisting of networked enterprise services are supported by interwork-enabling infrastructure services. Current business models, outsourcing, highly specialised core competencies and globalisation stress the openness and flexibility of enterprise computing environments. Openness provides possibilities for exploiting business opportunities, such as formation of collaborations between enterprises acting in different business domains or creating new kinds of business collaboration networks as a reaction to market changes. Agility is needed for tolerating and managing changes in both technology and business domains.

The architecture of enterprise computing systems must take into account the inherent autonomy-related properties of network business: differences in business strategies, operation models and legacy systems which induce technological and semantic heterogeneity between enterprise information systems. Enterprises are autonomous administration domains engaging practices best suited for their own needs and they have internal policies regarding the use of their services. Each enterprise exposes selected parts of their business functionality to clients as business services. The term business service is used to make a distinction between services provided by different vendors, public infrastructure (middleware) services and the more general notion of a service. Business services in enterprise computing environments are inherently dynamic entities since they are regulated by the business rules and policies of the corresponding enterprises. The behaviour of a business service provided by an enterprise may change without warning, therefore business services may become available while others may become inaccessible.

Due to the characteristics of modern business networks, interoperation of business services becomes problematic: interoperation must be reached simultaneously at different abstraction levels and between varying aspects of service-based communities [1]. The means to achieve interoperation must not conflict with the requirements and characteristics of modern electronic business networks. A "loosely coupled" model of interoperation can be established by relying on explicitly available meta-information describing the properties of business services and networks, interoperation-safe service discovery mechanisms, negotiable collaboration parameters and contract-based governance of the resulting business network [2, 3]. This kind of collaboration model based on the notion of business services, dynamically negotiable interoperation and eContracting is called a *federated service community*. Relying on the aforementioned mechanisms and service-oriented modelling of business networks, federated service communities tolerate the autonomy and dynamism inherent in inter-enterprise computing as well as mask heterogeneity of systems and processes.

Federated service communities require extensive infrastructure support for meta-information management and interoperability validation, community breeding, as well as runtime monitoring of communities. An essential part of the required infrastructure is the service-trading system, which is used for for retrieving suitable business services that implement desired business functionality. We use the term "service trading" to mean automated service discovery mechanisms and to make a distinction from more generic yellow-page services such as UDDI [4] which are more usable for service browsing. A service-trading infrastructure needs mechanisms to structure the amount of service offers into consistent hierarchies as well as means to guarantee interoperation between business services. These requirements can be fulfilled with service-typing.

A service type defines structural, semantic and behavioural characteristics for a certain kind of business services and provides the basis for interoperability validation through type checking procedures. Structural properties define the syntax for business service interfaces and exchanged documents. Semantic annotations can be attached to the documents. Behavioural descriptions define the externally visible behaviour of business services.

Based on the principles of type systems, object and component typing, and formal methods, a service type provides a rigorous basis for the tasks needed through the life-cycle of business services and networks. Interoperation of business services is provided by formally defined notions of substitutability and compatibility between corresponding service types. Type management infrastructure is a public and distributed meta-information management system which maintains service type information and relationships between service types. Type management infrastructure consists of public type repositories and name registries. Type repositories implement type checking and type matching functionalities which are needed for interoperation validation during service trading. Name registries are used for name-based resolution of meta-information in the open distributed service-architecture.

This paper presents an approach of service type management for collaborative computing environments which is realised in the web-Pilarcos project [2, 5]. Some of the application areas of service-typing illustrating its benefits during engineering of collaborative systems are described in Section 2. Section 3 describes characteristics of service types and type management systems. Implementation issues related to type repository functionality are introduced in Section 4. The service-typing model and type-management infrastructure are similar to the ODP model of open distributed processing [6, 7] and is applied to Web-Services–based environments.

2 Use of service types during business service life-cycle

Service types provide a uniform abstraction of business services for software engineers, enterprise modelling experts and system administrators to establish an interoperable collaborative system. Service types are used for interoperability validation during the design of business services and by code generation and conformance validation tools in implementation phase. For enterprise modelling service types provide means for interoperation verification between business services and networks. On deployment phase of business services and operation of business networks service types are used to establish conformance validation. The tools and platforms used in different phases use public infrastructure services through common middleware interfaces.

Service types are products of the software engineering design phase and their construction is supported by business service modelling tools. The design phase captures the characteristics of the business service that are determined by application domain, intended usage patterns and optionally by pre-existing service types. This interdependence is illustrated as a two-way meta-information exchange between desing-time service modelling tools and public type repositories in the Figure 1. Behavioural description provided by a service type can be utilised for simulation and prototyping purposes. When the business service designer is satisfied with the properties of the service type, it is published in a public type repository. Before the new service type can be published, it has to be checked against the typing rules of the targetted type repository. This functionality is to be embedded into the service modelling tools.

In the implementation phase published service types are used to aid business service development. Conformance and compatibility validations between a service type and the implementation interface are used to guarantee interoperability with eligible business services. Service types can be used as templates for the implementations. A suitable service type is fetched from a type reposi-

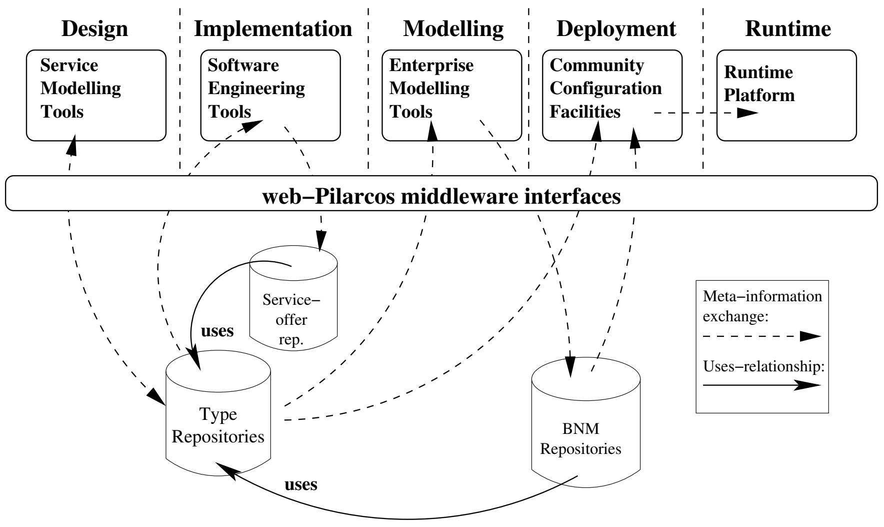

Fig. 1. Use of service types during business service life-cycle.

tory, after which generative methods can be used to provide skeleton code for the service implementation from the service type. This usage pattern of service types is similar to the MDA [8] approach; service types can be considered as a platform independent model (PIM) of business services. Service types could also be used to generate dummy implementations for testing the implementation with respect to other business services.

When the new business service is published in a public service offer repository, its behavioural properties and especially its conformance to the claimed service type are verified. Each service offer must correspond to a service type and when a service offer is exported to the system, its conformance with respect to the claimed service type is validated. Service offer repositories use type repositories to accomplish this task. This procedure guarantees type safety in them by ensuring that no ill-typed business services are published.

Service types are re-usable design components. Their primary purpose during the business network modelling phase is to provide well-structured design elements to be used with enterprise modelling tools. The structure and properties of a business network are modeled in Business Network Models (BNM). Further details of BNMs can be found from [5]. During the modelling phase, service types are composed into business roles. Business roles are attached to each other with connectors to describe the topology of the business network.

Enterprise modelling tools use type repositories for fetching interoperable service types. Type checking and matching operations provided by type repositories are exploited when verifying service type interoperability and searching for applicable service types to use in the business network model. The resulting business network model is published in a public BNM repository. During publication the corresponding business network model is verified using the validation functionalities provided by type repositories.

Two different kinds of activities can be identified from the deployment phase of federated service communities: community breeding and local computing infrastructure configuration. In the community breeding process a business network model is used as a community template to be populated with business services [2]. Service types are used for discovering the appropriate business services. Business services can be located from service offer repositories directly via their corresponding services types. However, the client is sometimes not interested in business services with a specific service type, but instead looks for business services which offer a certain kind of functionality. To serve this purpose, a service offer repository may exploit type matching functionality provided by the type repositories to return services of matching type. A matching business service provides the same functionality semantically, yet it is not exactly of the same service type.

As federated service communities are very loosely coupled open systems and their collaboration is based on negotiated contracts, runtime monitoring is needed for catching any erroneous behaviour and for establishing coordination of the community. For the local computation infrastructure configuration, service types provide the means for generating the monitoring components or configuring monitors with appropriate rules. Behavioural descriptions provided by service types are used as a specification which must be matched by the runtime behaviour of each business service.

3 Service types and their management

The main goal of type management functionality is to provide type safety. A type essentially enumerates properties for a kind of object: if two objects have equal types, then they should also have similar properties. Type safety basically means that entities labeled to carry a type are required to respect the properties subsumed by the type system. The type system provides rules for subsumed entity properties, and methods for checking that the entities managed actually fulfill these properties. For example, a programming language type gives requirements for the representation of data items, rules for possible operations on them, and rules for transformations between different types where meaningful. The validation of programming language type safety takes place at compilation time or in the interpretation process.

We apply the idea of typing and type-based validation of safety properties into the domain of business services. *Service types* are abstract descriptions which define the properties for a kind of business services. A service type defines the structural properties of a business service interface and documents. Semantic annotations can be attached to document descriptions to distinguish concepts that are semantically different but syntactically similar. In addition to the above static properties, service types consider the dynamic properties of business services in form of the behavioural descriptions. The behavioural description of a service type prescribes the externally visible behaviour to be expected from business services of the corresponding type. This approach is known as behavioural typing [9,10].

Service type safety essentially means that business service interoperation is guaranteed by service-typing rules. Interoperation of behaviourally typed artifacts reduces to two dual concepts of *substitutability* and *compatibility* [11]. Substitutability means that two business services can be replaced by each other. Compatibility means that two services can co-operate in a meaningful way if they are connected together: their interplay does not lead to a locking situation and the types (structure and semantics) of communicated messages are the expected ones. The concepts of substitutability and compatibility, and rules for verifying these properties have to be provided by the service-typing discipline. Substitutability is manifested in the service-typing system by type equivalence and subtyping rules. Compatibility of business service behaviour is provided with use of the notions of session type duality and subtyping [11].

Service type checking, that is procedure of verifying the correspondence between a claimed type and an entity, is performed during service publication. Given a service type and a service offer, a type checking algorithm verifies if the structure and behaviour of the service offer corresponds to the requirements of the service type. Functional properties of the type checking algorithm are characterised by the service-typing rules. Service type checking must be performed by the institution that is authoring the corresponding service offer repository before the service offer can be published.

To enable efficient service discovery mechanisms needed for service trading, the universe of service offers has to be structured into consistent, interoperation-preserving hierarchies. For example in CORBA [12] nominal subtyping hierarchies are used for structuring the service offers in traders. In UDDI [4] standardised taxonomies, such as North American Industry Classification System (NAICS), can be used to categorise service offers [13]. However, both of these aforementioned approaches are only suitable for (semi-)closed and regulated systems, where interoperation is manifested by the notions, naming conventions and concepts provided by out-of-band–negotiations. Service-typing discipline with similarity measures based on behavioural and structural properties of business services provides a more tractable structuring mechanism. For structuring purposes the type system for service types must have concepts of type equivalence and subtyping, and algorithms for checking these relationships between service types.

Type management infrastructure is needed for guaranteeing type safety during service trading in collaborative systems. Type management infrastructure consists of two kinds of infrastructure services: type repositories and name registries. Type repositories are persistent storages of meta-information, in this case service-typing information. Name registries provide facilities for name-based resolution of meta-information and are needed for governing naming policies. In the following the characteristics of type repository functionality are described.

4 Type repository services

A type repository provides storage and management for service type descriptions, relationships, and naming of types. A type repository is basically a persistent storage of meta-information. The meta-information consists of type system description, type descriptions and associations between these descriptions. The type repository interface provides operations for adding, removing and querying type information. The characterizing features of the type repository include information representation method, type checking procedures, type matching procedures, type naming conventions, and replication/partitioning for cooperating type repositories [6, 14, 15]. The information represented consist of service types, document types, basic types and relationships between the corresponding types.

Service types are represented using a XML-based description language. A service type schema comprises definitions for service attributes, document types, and an interface protocol. Service attributes declare additional properties needed for service selection. Document types define the structure for exchanged messages using XML-Schemas [16]. Interface protocol provides a description of the the externally observable service behaviour. A simplified abstract syntax of the service type schema is given in Figure 2.

```
<serviceType
  name = xs:string>
(serviceAttributes?,
 documentTypes, iproto )
</serviceType>

<serviceAttributes>
 (property+)
</serviceAttributes>

<property
  name = xs:string
  type = xs:QName
  concept = xs:QName?>
constraint?
</property>

<documentTypes>
 [XML-Schemas for documents]
</documentTypes>
```

```
<iproto
  name = xs:NCName>
(channel+, session)
</iproto>

<channel
 name = xs:NCName>
 [channel properties]
</channel>

<session
  name = xs:NCName>
(parallel | sequence |
       if | repeat | choice )?
</session>
```

Fig. 2. Simplified schema of a service type description.

Service attributes, such as quality of service prerequisites, are provided by `serviceAttributes`-element which includes a set of `property`-elements. Each property element defines a named property with a type, a semantic concept from some shared ontology and an optional constraint. A constraint can be a set or range-based restriction over the value-space of the property's type. For example a property of type `xs:int` can be attached with a range-based constraint such that the integer value must be between values 0 to 5.

Document types used by the service type are defined in the `documentTypes`-element. New documents types can be defined using the constructs represented by XML-Schema –language [16]. Existing document type definitions can be reused using namespace import mechanisms. Semantic annotations can be attached into document types when needed.

An interface protocol described by the `iproto`-element in Figure 2 defines bilateral behaviour for a business service. By constraining service to bilateral connections, the modularity and independence of each business service and thus reuse possibilities are increased. The interface protocol declares the service behaviour (`session`-element) and a set of communication channel definitions (`channel`-element). Sequences of communication actions, choices, parallel processes, conditional processes and repetition are used for describing the service behaviour. A session does not describe the whole life-time behaviour of a service but one service conversation-instance with a client. Parallel instantiations of service sessions with several clients are thus not modeled. Channel declarations introduce communication channels to be used for interacting with the business service.

Engineering level information, such as binding of a service instance into a specific communication protocol or address, is not covered by a service type. Instead this kind of technical information as well as values or value ranges for service attributes are declared in *service offers*. A service offer provides the bindings between the communication actions defined in the service type and the corresponding Web-Services–compliant business service interface. These technical bindings can be given for both SOAP/WSDL [17] and REST (REpresentational State Transfer) [18] based business services; the communication mechanism is not constrained by the service type.

In service type checking, the structural and behavioural properties of a service offer are verified against the properties defined by a service type. Concerning behaviour, each communication action described in the service offer has to be matched with the communication behaviour defined in the service type. Both substitutability and compatibility relationships can be used as a behavioural matching criteria. For structural properties, similarity between the document structures used in the service offer and document types declared in the service type must be matched.

Thus, service type checking procedure requires thus algorithms for verifying behavioural substitutability and compatiblity between the behaviours represented by a service offer and service type. Algorithms for verifying subschema relationships between the corresponding document types are also needed. Algorithms for checking behavioural substitutability and compatibility are based on the behavioural subtyping relation and notion of behavioural duality of the session typing discipline [11, 19]. These syntax-directed algorithms are expected to be efficient in practise [11]. For the purpose of subschema checking, XML-document structures are to be considered as regular tree grammars and the simulation relation between corresponding automata is used as a subschema relation [20]. In general case the algorithm for deciding the subschema relationship has exponential complexity in the size of the schema, althoough the complexity can be decreased to polynomial by constraining the schemas to be labelled-determined [21]. Semantic correspondences between the concepts used in the exchanged documents are not validated during the previous verification procedure. Instead, semantic “proof obligations” are gathered during type checking from the semantic annotations in the document structures. These proof obligations can then be validated utilising standard reasoning techniques and tools used for ontology management.

Type matching procedures search for a set of types that satisfy the given relationships with given types. Type matching is for searching for example service types that are substitutable or compatible with each other. Type matching is used during business service discovery to provide more flexibility and wider range of results. Type matching relationships (substitutability, compatibility) are asserted by users and validated by the corresponding type repositories. Thus no special algorithms are needed for validating or inferring type matching except for the substitutability or compatibility algorithms needed during validation of the match. A user may also proclaim that two service types match via adaption. In this case the user must provide the appropriate adaption service or adaption specification. These kinds of matches can not be verified. The form of appropriate adaption specifications are to be studied.

The type naming conventions are required for maintenance of identified service types in the repository. In web-Pilarcos environment type names are formatted as Uniform Resource Identifiers, or URIs [22]. When type descriptions are stored into repositories they are given unambiguous names. Names are maintained in the name registry subsystem which is responsible for name-based resolution of meta-information. Name registration must be strictly controlled by name registries to maintain unambiguousness and global uniqueness of names. A hierarchical name registry system with naming domains based on the namespace structure is one possibility to establish these requirements. Different kinds of namespace structures can be established for example on the basis of application, business or organisational domains. Aliasing mechanisms provided by names are used to refer to a type description in another naming domain via a local name.

In order to provide an useful infrastructure service for publishing and relating service types for an open network, it is necessary to arrange for collaboration between multiple repositories. Replication of type repository functionality and local caching of type information are utilised to increase efficiency of the system. Document type and service types declarations are reused via importing namespaces or referencing names from other namespaces in service types and business network models. Imports and references are considered as read-only references and the imported type definitions can not be modified or extended. Circular dependencies between type descriptions are not allowed or supported. Subtyping by extension is allowed during publication of new document and service types. When extending a document type with new structures or a service type with new behaviour, the resulting subtype must be published in the same repository as the corresponding super-type. These restrictions simplify the maintenance of typing hierarchies and their consistency. Evolution of type descriptions necessitates versioning support in the type repositories. Maintenance of type description consistency evolution requires mechanisms similar as those taken in the KAON platform [23] for example.

Service-typing discipline provides rules for expressing properties of business services via service types and hierarchies between them. Essentially, service typing thus provides a kind of a distributed, shared ontology which defines rules to express terms and relationships between the terms. However the "ontology" induced by a typing discipline has some differences when compared to ontologies used typically for knowledge representation. Typical knowledge representation logics and languages are more suitable for representing static properties since they usually do not have concepts for expressing communication and concurrency. Service types are software engineering artifacts whereas ontologies are knowledge engineering artifacts: ontologies are by nature descriptive whereas service types are declarative. Ontologies are usually used as annotated auxiliary information whereas types describe more fundamental properties and are more tightly bound to structural and behavioural properties of business services. Performance-wise, type checking is syntax directed and can use more specified proof-techniques whereas inference in logic languages has to use more general, usually more complex techniques.

The web-Pilarcos type repository provides its functionality through Web-Services– compliant interfaces. The type repository functionality is implemented with Java, as it currently provides the most comprehensive support for XML processing and Web-Services–technology. The XSB logic programming and deductive database system [24] is used to implement part of the back-end functionality and especially to infer the transitive closures induced by typing relations. The type checking functionality will be implemented using an ML-language because of its suitability for processing term-like structures and large code-base related to functionalities needed for verifying properties of service types.

The type repository functionality of the web-Pilarcos platform is similar to the type repository function of ODP reference model [7] and Meta-Object Facility standard of OMG [25]. The actual data or information (M0-level of MOF) in the web-Pilarcos platform consists of service offers. The model-layer (M1) consists of different service-types. The meta-model layer (MOF M2-level) describes target concepts and type relationships, such as the notions of service types and compatibility relationships. It should be noted that basically only one meta-model is currently provided in web-Pilarcos: the one based on the notion of service types. The meta-meta-model (M3) contains hard-wired basic concepts such as meta-classes, meta-associations and data types of XML-Schema.

5 Conclusion

Service-typing and corresponding type management infrastructure provides a valuable tool for service-oriented software engineering. A service type provides an unifying abstraction of business service functionality to be used by software engineers, enterprise modelling experts and enterprise system administrators through the life-cycle of business services. Type management infrastructure consists of public type repositories and name registries. Type repositories are persistent storages of typing information whereas name registries provide name-based resolution of meta-information and control over naming conventions.

Because collaborative systems involve interaction between independently administered services, the supporting infrastructure must provide service trading with type safety properties. In this context, type safety involves much of the same issues as interoperability of business services. Having been involved with the development of the ODP trading function, and type repository standards – mostly shared publications with OMG [6, 26] – we consider that the ODP reference model provides a necessary element for interoperability facilities. However, the Web-Services technology family lacks similar consistency in type management issues. The web-Pilarcos type management system provides enhancements on both these frameworks: the ODP type system is complemented with more specific meta-information needed on business services and behavioural aspects of services. In comparison to the Web-Services approach, we suggest controlled service type management within the infrastructure layer, and increased amount of in-band regulation of interoperation knowledge and naming conventions.

Acknowledgement

This article is based on work performed in the Pilarcos and web-Pilarcos projects at the Department of Computer Science at the University of Helsinki. The Pilarcos project was funded by the National Technology Agency TEKES in Finland, Nokia, SysOpen and Tellabs. In web-Pilarcos, active partners have been VTT, Elisa and SysOpen. The work much integrates with RM-ODP standards work, and recently has found an interesting context in INTEROP NoE collaboration.

References

1. Ruokolainen, T., Kutvonen, L.: Interoperability in Service-based Communities. In Bussler, C., ed.: Business Process Management – BPM 2005 Workshops, Springer-Verlag (2005) To appear.
2. Kutvonen, L., Metso, J., Ruokolainen, T.: Inter-enterprise collaboration management in dynamic business networks. In: OTM Confederated International Conferences. Volume 3760 of LNCS., Springer-Verlag (2005)
3. Metso, J., Kutvonen, L.: Managing Virtual Organizations with Contracts. In: Workshop on Contract Architectures and Languages (CoALa2005), Enschede, The Netherlands (2005) To appear in the IEEE Digital Library.
4. UDDI: Universal Description, Discovery, and Integration of Business for the Web. (2001)
5. Kutvonen, L., Ruokolainen, T., Metso, J., Haataja, J.: Interoperability middleware for federated enterprise applications in web-Pilarcos. In: Interoperability of Enterprise Software and Applications, Springer-Verlag (2005)

6. ISO/IEC JTC1/SC7: ISO/IEC 14769: Information technology - Open distributed processing - Type repository function. (1999)
7. ISO/IEC JTC1/SC7: ISO/IEC 10746: Information technology – Open Distributed Processing – Reference model. (1998)
8. Frankel, D.S.: Model Driven Architecture: Applying MDA to Enterprise Computing. OMG Press (2003)
9. Ravara, A., Vasconcelos, V.T.: Typing non-uniform concurrent objects. In: CONCUR '00: Proceedings of the 11th International Conference on Concurrency Theory, London, UK, Springer-Verlag (2000) 474–488
10. Chaki, S., Rajamani, S.K., Rehof, J.: Types as models: model checking message-passing programs. In: POPL '02: Proceedings of the 29th symposium on Principles of programming languages, New York, NY, USA, ACM Press (2002) 45–57
11. Vallecillo, A., Vasconcelos, V.T., Ravara, A.: Typing the behavior of objects and components using session types. Electronic Notes in Theoretical Computer Science **68** (2003) Presented at FOCLASA'02.
12. Object Management Group: Common Object Request Broker Architecture (CORBA) v3.0.3. (2004) OMG Document formal/04-03-01.
13. Singh, M.P., Huhns, M.N.: Service-Oriented Computing: Semantics, Processes, Agents. John Wiley & Sons, Ltd. (2005)
14. Muenke, M., Lamersdorf, W., Christiansen, B.O., Mueller-Jones, K.: Type management: A key to software reuse in open distributed systems. In: 1st International Enterprise Distributed Object Computing Conference (EDOC '97). (1997)
15. Kutvonen, L.: Trading services in open distributed environments. PhD thesis, University of Helsinki (1998)
16. W3C: XML Schema Documentation; Part 1:Structures, Part 2: Datatypes. 2nd edn. (2004) W3C Recommendation, `http://www.w3.org/XML/Schema`.
17. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. W3C. 1.1 edn. (2001)
18. Goth, G.: Critics Say Web Services Need a REST. IEEE Distributed Systems Online **5** (2004) 1
19. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Proceedings of the 7th European Symposium on Programming, Springer-Verlag (1998) 122–138
20. Amadio, R.M., Cardelli, L.: Subtyping recursive types. ACM Transactions on Programming Languages and Systems **15** (1993) 575–631
21. Brown, A., Laneve, C., Meredith, G.: PiDuce: A process calculus with native XML datatypes. In: 2nd International Workshop on Web Services and Formal Methods (WS-FM 2005), Springer-Verlag (2005) To appear.
22. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax (1998) RFC 2396.
23. Maedche, A., Motik, B., Stojanovic, L.: Managing multiple and distributed ontologies on the Semantic Web. The VLDB Journal **12** (2003) 286–302
24. Sagonas, K., Swift, T., Warren, D.S.: XSB as an efficient deductive database engine. In: SIGMOD '94: International conference on Management of data, New York, NY, USA, ACM Press (1994) 442–453
25. Object Management Group: Meta Object Facility (MOF) 2.0 Core Specification. 2.0 edn. (2003)
26. ISO/IEC JTC1: ISO/IEC 13235: Information Technology – Open Distributed Processing. ODP Trading function. (1997)