Enhanced trading service in middleware for inter-organisational applications

Markku Vähäaho, Lea Kutvonen

University of Helsinki, Finland
(Markku.Vahaaho | Lea.Kutvonen)@cs.Helsinki.FI,
WWW home page: http://cs.Helsinki.FI/

Abstract. Current information processing needs of companies require inter-organisational cooperation. New middleware is expected to provide facilities for capturing the workflow, searching for available members for the workflow, and ensuring that a suggested set of members is able to interoperate both in semantical terms and technically. The Pilarcos project has developed a prototype middleware where inter-organisational application management is based on explicitly stated, platform-independent, multilateral contracts that define the forms of cooperation between performing components. This paper discusses the Pilarcos trader prototype as a federation contract providing service. The Pilarcos middleware feasibility is much dependent on the scalability of Pilarcos trading, but the measurements show that the algorithm behaves well under changes in service offer space and business architecture complexity.

Keywords:
trading service, middleware, federation, application management, inter-organisational cooperation, architecture descriptions

1 Introduction

Current information processing needs of companies require inter-organisational cooperation. Consequently, new protocols and architecture models are built for using open services available through Internet [1–4]. Development efforts are directed to the way services and conversations between service providers should be described and formulated.

New middleware is expected to provide dynamic and automatically controlled facilities for inter-organisational cooperation. Current middleware solutions already provide reasonable support for managing the technological heterogeneity of operating systems and network solutions and for adapting to dynamic changes in available resources. However, in a multi-organisational environment, decisions on provision of application-level service, on operational policies, on platform architectures, and on communication protocols can be done independently from other systems. Further difficulties encountered by inter-organisational applications include the need to adapt to the constant change in potential partners and the independently driven development of services in each of these systems. These challenges should be addressed by improved middleware support. Facilities are needed for expressing service requirements, and for describing alternative conversation patterns between services. Furthermore, facilities are needed for negotiating joint rules on new cooperation relationships.

Therefore, new middleware should provide facilities for capturing the workflow, searching for available members for the workflow, and ensuring that a suggested set of members is able to interoperate both in semantically and technically. The Pilarcos project (Production and Integration of Large Component Systems) at the University of Helsinki studies these needs as part of a larger set of cooperational management services across organisations and platforms. The project has developed a prototype middleware described below.

In the Pilarcos model, the inter-organisational application management is based on explicitly stated, platform-independent, multilateral contracts that define the forms of cooperation between performing components [5]. Federation establishment focuses on capturing a shared understanding of business logic and semantics of the services exchanged, but the selection of members is further restricted by the technical communication requirements of each member. Federation contracts are structured according to a business architecture where a set of roles denote the services expected from member components. Membership criteria includes conformance to the required service in the business architecture and technical interoperability with the other members in the federation. Member components are selected from service offers exported to trading services.

This paper discusses the Pilarcos trader prototype as a federation contract providing service. Because of its central role in Pilarcos middleware, the performance profile of the Pilarcos trader is essential for the feasibility of the middleware services. First, Section 2 outlines the overall Pilarcos middleware in order to give reference to the use of federation contracts. Section 3 focuses on the enhanced trading mechanism. Section 4 includes measurement results of the population process and show that the major cost of the Pilarcos middleware is not in the combinatory population algorithm. Section 5 concludes the presentation with future work, related work and evaluation.

2 Pilarcos middleware

Pilarcos middleware provides application programmers with pervasive, platform-independent tools to manage federations. However, the use of the functions remains explicit. In this section we review the concepts provided for the programmers and the middleware functions available for building applications [6].

The key concept for inter-organisational cooperation is that of federation. Federation is an identified and structured collaboration of a group of partner components in peer-to-peer communication relationships. Each federation is maintained by a federation contract. A *federation contract* captures the structure of the federation by reference to a business architecture, the selected members of the federation or selection rules for membership, and policy decisions agreed for the particular federation.

A *business architecture* is defined by a set of roles, interactions between roles and a set of policies. The business architecture description does not fix the identities of the participating systems. Instead, roles are associated with *service type* that defines the the class of service required. Members for the federation are selected based on service offers exported to trading services. The service offers include metainformation on component's service type, technology requirements, conversation protocols expected, operational policies, cost, location, etc.

Policies in a business architecture have two targets. First, a policy rule can be set to govern the behaviour of a component in a role. For example, different information retrieval strategies can be preferred depending whether there is need to save space or time in a search. This kind of expectation can be passed on to a component via a role related policy. Second, a policy rule can be set to govern interactions in the federation. The business architecture can model alternative interaction sequences and the policy value can be used to determine which of the alternatives should be used for the federation.

Component programmers provided with a repository from where business architectures can be retrieved at need. There are separate processes for designing and publishing business architectures for a different group of practitioners. The business architectures are considered to be globally available and interpretable via federated repositories. Service types related to roles can be interpreted also as requirement definition – besides selection criteria for components.

In the Pilarcos project, no new software production tools are provided. However, we see the recent development of OMG MDA (model driven architecture) [7] tool chain a complementing approach. Here, a component is to be understood loosely a service implementation encapsulated in such a way that platform services are able to manage its lifecycle (deployment, instantiation, termination, activation and deactivation). Although Pilarcos project has used component based platforms (OpenCCM, MicoCCM, JBoss) for experimenting and prototyping, the Pilarcos architecture does not require component techniques to be used.

Requirements for components cover two aspects. First, the components are expected incorporate Pilarcos federation management interface. This interface defines operations for policy manipulation, request for federation establishment or termination. Second, the components are expected to follow policy rules stated in the local policy repository.

The Pilarcos management services enhanced trading, federation management, type management, federated binding, policy repositories follow RM-ODP model of middleware. The RM-ODP standard by ISO and ITU defines – in addition to terminology and viewpoints – a middleware model ([8], clauses on engineering viewpoint, functions and transparencies) that can be used to support inter-organisational applications.

The enhanced Pilarcos trader provides two main operations: exporting a *service offer* for a specific service type (`export`), and populating a business architecture with mutually compatible service offers (`populate`). The former operation is used by an administrative tool used by a service provider. The latter operation is used by the Pilarcos federation manager on behalf of the application wishing to establish a federation. The `populate` operation takes an incomplete federation offer as a parameter, and returns one or more completed federation offers. No separate constraint parameter is used; instead, the incomplete federation offer typically contains a *pre-filled* service offer for the populating role itself, defining its policies for the federation. Thus, the population process is completely symmetrical: any role that has been left empty in the incomplete federation offer is populated by the Pilarcos trader. This makes it easy to do partial repopulations for failure recovery or adaptation purposes.

The federation managers are responsible of running the protocol for negotiating, maintaining and renegotiating federation contracts. For federation managers, the essential information element is of type *federation offer*. It is a combination of compat-

ible service offers, one for each role in a specific business architecture. The federation establishment protocol is initiated by a client request. As a first step, a service offer that describes the client itself is positioned into a federation offer element. The Pilarcos trader then populates the rest of the roles. As a result, several suggested federation offers are returned for the client to choose from.

For testing whether a component is suitable for a federation, service type matching is needed. The Pilarcos middleware design includes an enhanced version of ODP type repository [9] for holding relationship information between generic types (service types, binding types, interface types) that are technology-independent and used for matching purposes and technology-dependent templates that are used for instantiating the corresponding components and objects. This mapping information is created by system programmers separately from business architecture descriptions and service offers.

Pilarcos middleware expects that components can be managed by policies, much like in policy-based management systems (e.g. [10]). Administrators can create and set policies for component groups. Furthermore, federation contracts are stored into the policy repository and thus federation contracts become an integral part of component management mechanisms. The current implementation is bare.

Federation contracts are formed using platform-independent models. In order to realise federation management events, the abstract notations must supported with mappings onto technical realisations and service instances. For example, platform-independent requests for deployment, instantiation or binding need to be mapped on platform-specific installation scripts and factory services. Especially, federated binding requires both contractual and actual modes and transitions between the modes, as described in ODP binding framework [11].

The Pilarcos approach uses two types of domains: administrative domains and technology domains. An administrative domain can be for example an organisation, a company or a department with authority to do independent operational decisions about the way it runs its business. Organisation-wide policy management is needed to allow IT-system administrators to reflect operational policies – such as restrictions to cooperation partners, payment related conversation styles and time of availability of offered services – consistently onto all applications of the organisation in an automated manner. Service descriptors, service management rules, and policies are defined at the administrative domain level in technology-independent terms.

Component management rise the need of separation between technology domains. For each technology, there are various facilities for deployment, instantiation and termination of components. A technology domain is here limited within an administrative domain for simplicity. At each technology domain, service descriptors and service management rules are mapped onto technical engineering solutions. Naturally, these mappings follow a pattern common to all administrative domains.

The Pilarcos middleware services reflect the administrative and technology domains and the need to cooperate across the domain boundaries [6]. There are collaborative middleware services with a running agent at each administrative domain for negotiating federations and advertising available services. These agents take care of making requests to their peers at other domains, as there is no authority to otherwise invoke management actions at a foreign domain [12]. The requests carry contracts to pass relevant meta-information that identifies what should be done and how. On

the other hand, there are local management facilities running at each technology domain for managing components. In addition, at each administrative domain there are agents responsible of mapping abstract federation contract information into a form usable at technology domains.

3 Enhanced Pilarcos trader

The Pilarcos trader provides an enhanced trading service by populating a business architecture with service offers with mutual interdependencies. Before the population process algorithm can be understood, a modeling example is used to leads to a set of basic information element definitions.

3.1 Modelling with business architectures

As an example of an inter-organisational application, we use a hypothetical tourist service. The tourist service provides travel information, hotel booking and car rentals, typically for a specific area such as a city. The tourist has a travel planning application on her PC or mobile device, which remotely connects to a tourist service. Several competing tourist service providers may exist for an area, with a variety of features and prices. Payment for the service is done electronically, mediated by a trusted payment service, which is typically provided by a bank or a credit card company. In this scenario, payment services also compete for customers by price, credit and other factors. Not all payment services and methods can be used with all tourist services. The travel planning application must find a suitable tourist service for the area the user is interested in. In addition, we do not wish to unnecessarily restrict payment options by fixing the payment service in advance, but to choose it according to the situation instead.

In the example, choices for the tourist service and the payment service are not independent. All of the participating systems must agree on the workflow for secure payment to be possible; they must use common communication protocols; and typically an established trust relationship has to exist between the payer and the mediator as well as between the mediator and the payee. Using just, say, trading service for locating suitable tourist service providers and suitable payment services would require the application to manage variations in provision of both services and intelligence for ensuring interoperability.

In the example application, the community formed by the tourist planner, the tourist service and the payment service is modelled as an explicit business architecture description. The roles of the tourist service and the payment service need to be populated by the Pilarcos trader with services compatible with each other and the tourist planner. A role is only a placeholder for an implementing component, and is described by an associated service type.

In Pilarcos middleware, a service type defines a class of services that have the same logical functionality but may differ in implementation. A similar concept is used in the joint ODP/OMG Trading Service standard [13] and the UDDI repository standard [14]. In our model, a service type defines the set of abstract interfaces that the service exposes, along with federation policy frameworks for parameterising the

behaviour at the interfaces. A service type can be reused in multiple business architectures; a service provider need not be aware of the business architectures that the service type may be part of.

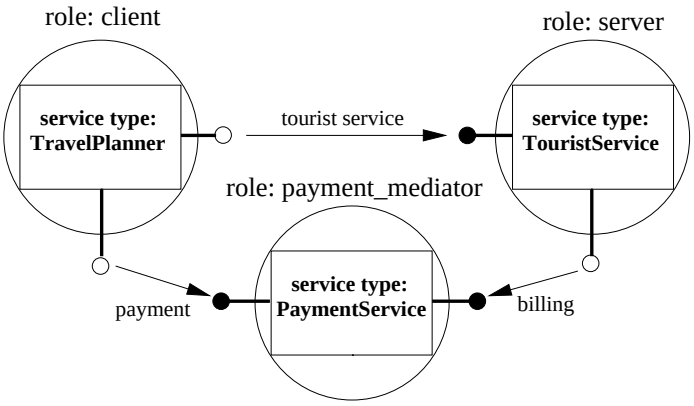

Fig. 1. Business architecture for the tourist service example.

In addition to roles, business architecture descriptions define interaction relationships and dependencies between the participants, expressed as bindings. For example, the example description shows that using a tourist service generates a bill that needs to be settled with a payment service. For this interaction sequence, two alternatives are modeled: prepayment and billing afterwards. Which pattern is actually used is determined by a policy selection in the federation contract; the policy decision is done by middleware services based on the acceptable policies announced by each potential participant.

Technically, bindings between the roles represent both communication channels and other dependencies between components. They link the interfaces of the roles together, creating compatibility requirements for the roles. For an illustration of the relation between service types, interfaces and roles, refer to Figure 1. In the figure, service types are drawn as boxes, roles as circles, and provided and required interfaces as small filled and hollow circles, respectively.

3.2 Federation policies and policy frameworks

In the CORBA Trading Service standard, services are described by their properties, coded as name-value-pairs. In the Pilarcos model, *federation policies* are used to describe service properties and behaviour. Federation policies, or just *policies* for short, are also coded as name-value pairs. Although policies may carry both business policies and technical property information, the Pilarcos trader makes no distinction between the different kinds of policies.

Related policies are collected together as *policy frameworks*. Because a policy framework, together with a description of its semantics, defines an ontology for policies, policy frameworks are subject to publication and standardisation. The structure

of a policy framework is defined by its *policy framework type*, which is a named set of *policy types*. A policy type defines the name and, in our implementation, the OMG IDL type of the policy. Any OMG IDL type, such as integer, string or structure, may be used as a policy value type. Comparing policy frameworks, which must be of the same type, is done by comparing the individual policies within the frameworks. Policies with equal values are defined to be compatible.

The Pilarcos services also recognise special policy value types that make it possible to use simple constraints as policies. Currently, special types are defined for expressing simple closed intervals of integers and real numbers, for example $[1, 5]$, and string set constraints. These types of policies are compatible if their intersections are non-empty.

A string set constraint policy consists of a constraint type and a set of string values. For example, a string set constraint policy of the form `protocol = one_of` {"`IIOP-1.1`", "`IIOP-1.2`", "`RMI-1.0`"} requires that exactly one of the listed protocols must be used as the final protocol. Three constraint types can be used. In increasing order of strength, they are

- `some_of`, which requires that one or more of the strings be present in the final policy;
- `one_of`, which requires the final policy to contain exactly one of the strings; and
- `exactly`, which requires that the final policy must contain exactly the original strings.

All constraint types are exclusive: they prohibit additional values.

The intersection of two string set constraint policies is a string set constraint policy whose constraint type is the stronger of the original types, and whose set of strings is the intersection of the original sets. For example, the intersection of `some_of` A, B, C and `one_of` B, C, D becomes `one_of` B, C. In the case of an `exactly` policy, the intersection is empty unless the intersection of the string sets is equal to the string set of the `exactly` policy. Note that it would be possible to have other types than strings as values with the same rules; strings were chosen for simplicity.

Intervals and string set constraints are sufficient for expressing most common constraints, and yet are simple enough for efficient calculation. Most importantly, they have the property that calculating their intersections is an associative and commutative operation, which allows the Pilarcos trader to perform the calculations in any order and optimise the search process.

3.3 Interface types

The Pilarcos model has two layers of interface types: abstract and concrete. An abstract interface type represents a logical functionality, defined by an either formal or informal description; a concrete interface type defines the actual operations supported in a concrete interface definition language, such as OMG IDL. A single abstract interface can be implemented as multiple concrete interfaces. The distinction is similar to that between the computational and engineering viewpoints in the ODP Reference Model [15].

Abstract interface types are platform independent; concrete types are platform specific. This two-layered model is necessary to support federation of sovereign systems [16, 6.2], which may use differing implementation technologies. Interfaces are considered compatible if their concrete types are the same, or if an interceptor (adapter

or bridge) is available for differing types. Interface types, interceptors and their relations are registered in the type repository, which provides type matching operations. The prototype implementations of the Pilarcos services code concrete interface types as strings, and the type repository simply compares them for equality.

The use of abstract service types is designed to directly support more complex interoperability tests, especially semantic matching or protocol based matching. A variety of existing research elsewhere can be combined in Pilarcos context [17]. Currently only platform related differences or simple application interface differences can be controlled. Also, type repository administrator tools are missing.

3.4 Service types

A service type definition consists of a set of required and provided abstract interfaces, and a set of policy frameworks attached to the interfaces. Defining required interfaces in addition to provided interfaces makes service types abstract, composable components [18].

Figure 2 is a condensed pseudocode example of the tourist service type; types are written with capitalised initial letters. Compare this with Figure 1.

```
service type: TouristService
{
    provides interface: TouristServiceInterface tourist_service_i;
    requires interface: BillingInterface        billing_i;

    policy framework: TouristServicePolicies tourist_service_pf
        attached to interface: tourist_service_i;
    policy framework: PaymentPolicies        payment_pf
        attached to interface: billing_i;
}
```

Fig. 2. Pseudocode example of a service type definition.

The policy frameworks attached to an interface parameterise the behaviour associated with the interface. Although not visible from the example, it is possible for a single policy framework to be attached to more than one interface, which implies that the policies are shared. In the tourist service example, the payment service has a shared policy framework for both the billing and payment interfaces. This ensures that the payment service, the biller, and the payer have compatible payment policies.

3.5 Business architectures

A business architecture definition consists of roles and bindings between the roles. To the Pilarcos trader, a role is a named instance of a service type; the interfaces of the service type are also the interfaces of the role. Bindings connect a required interface of one role to a provided interface of the same abstract type in another role. This model

bears a close resemblance to architecture description languages [19], but extends them by supporting run time population of the roles.

Service types, and thus also services, can exist independently of business architectures, which facilitates reuse of both service type definitions and the implementing components. Defined like this, service types are akin to the configuration-independent component definitions of the Wright architecture description language [20]. Wright also incorporates formal methods for behavioural compatibility verification; similar methods could also be used to verify business architectures in Pilarcos.

When interfaces of two roles are connected by a binding, policy frameworks of the same type attached to the interfaces are also implicitly connected. For service offers for the roles to be compatible, their connected policy frameworks must then also be compatible. The compatibility requirement can even extend over several roles via service types that have shared policy frameworks, as in the tourist service example where payment policies are shared by all roles.

In the general case, the implicit connections between policy frameworks form undirected graphs, called *policy framework graphs*, with roles as vertices and bindings as edges. Each policy framework graph represents a policy framework implicitly shared between the roles in the graph. When a business architecture is registered in the Pilarcos type repository, it finds the policy framework graphs of the architecture by a simple depth-first algorithm and stores them for later use by the Pilarcos trader.

3.6 Service and federation offers

A service offer describes a concrete service of a specific service type. In consists of concrete interface types and interface references for the abstract interfaces of the service type, as well as policy values for the policy frameworks of the service type.

A federation offer describes an entire community, whose structure is defined by its business architecture definition. It consists of one service offer for each role in the architecture. If one or more service offers are missing, the federation offer is *incomplete*, otherwise it is *complete*.

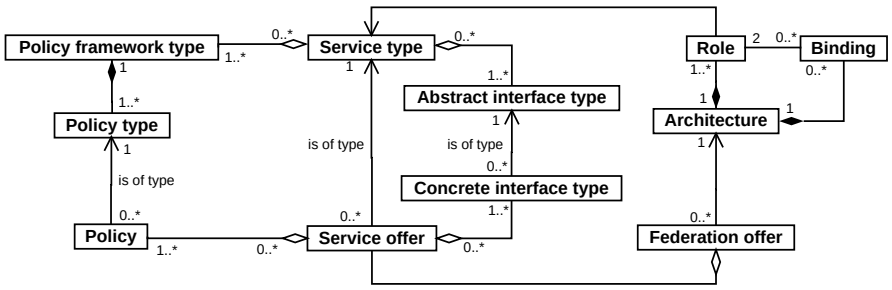

Fig. 3. UML class diagram of Pilarcos trading concepts.

The Pilarcos trading concepts form two layers, describing types and implementations respectively. This is illustrated in Figure 3 with types in the upper part of

the diagram and the corresponding concrete entities in the lower part. All types are implementation independent and form a shared ontology as basis for federation.

3.7 Search algorithm

In the population process, the Pilarcos trader searches its database for compatible service offer combinations. Due to the combinatorial nature of the problem, the number of possible service offer combinations can be very large. In the general case the problem of whether complete federation offers exist for a given architecture and a given offer database is NP-complete (CLIQUE reduces to this problem in polynomial time). Yet the population process should be relatively quick to be practical. This is achieved by two means: using an optimised search algorithm and restricting the extent of the search.

The populator can restrict the search by giving two limits: the maximum number of federation offers to be returned, and the maximum duration for the search process within the trader. The search terminates when either limit has been reached or the entire offer database has been searched without success. In most cases, returning one or a couple of federation offers suffices.

The search algorithm proceeds depth-first with respect to the roles of the architecture in order to find the first complete federation offers as quickly as possible. This avoids having to search the entire offer space. Each role has an associated service type; since the trader indexes service offers by their service type, the candidate offers for each role can be accessed rapidly. Pre-filled roles are handled just like other roles, except that they only have one candidate offer.

To reduce the size of the search tree, the Pilarcos trader sorts the roles into ascending order according to the number of candidate service offers. The search algorithm visits the offers in the roles recursively, beginning with the first role. The interfaces and policies of a candidate offer for the current role are compared with those of the offers selected for the previous roles; if they are compatible, the search proceeds to the next role, otherwise the next candidate offer for the role is tried until none are left. When a compatible offer is found for the last role, the selected offers form a complete federation offer.

Compatibility between service offers is determined by interface and policy compatibility of the offers. Interfaces are compared by querying the type repository for interface compatibility. This is done only once per binding in the architecture. Policy compatibility requires that the policies within a policy framework graph are compatible. Compatibility rules for different policy types were discussed earlier in Section 3.2.

The number of interfaces per service type is typically small, so comparing them does not pose a significant overhead. The number of policies in a service offer can be much larger; therefore it is important to minimise the number of policy comparisons. This is done by taking advantage of policy framework graph data provided by the type repository.

At the beginning of the search process, the Pilarcos trader requests both the architecture definition and the accompanying policy framework graph data from the type repository. It then allocates space for one *graph-specific policy framework* per each graph. For the tourist service example, graph-specific policy frameworks are illustrated in Figure 4; policy framework types are marked “Tou” and “Pay”. Policy

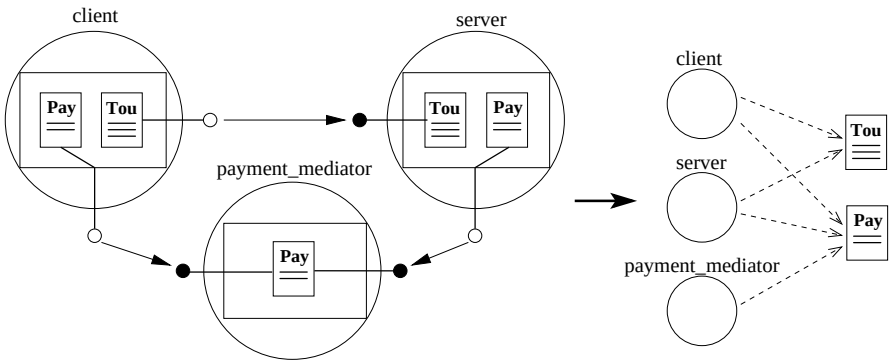

Fig. 4. Graph-specific policy frameworks in the example architecture.

comparisons are done by calculating the intersections of the policies in the candidate offer with those in the graph-specific policy frameworks. If the intersections are non-empty, the offer matches, and the intersections are stored in the graph-specific policy frameworks. Since the algorithm is recursive, each invocation has its own copy of the graph-specific frameworks, containing the intersections of the policies in the previously selected offers. This minimises the number of calculations needed.

Because the federation offers returned by the Pilarcos trader are to be used as basis for federation establishment, where only the compatible subset of interval and string set constraint policies can be used, the Pilarcos trader replaces the original policy values with their intersections before returning the results. In the present algorithm, when a complete federation offer has been found, intersections of policies for it are already available and can be written over the originals in the federation offer. The entire algorithm is outlined in Figure 5.

As presented here, the search algorithm performs the least possible number of comparisons, and uses space only in linear proportion to the number of roles and the number of policy framework graphs. This yields good results in practice, as will be seen in the following section.

From a combinatorial standpoint, each newly formed federation offer should be adequately different from the earlier so that the populator does not receive almost identical offers. Furthermore, the service offer space should be covered uniformly instead of always returning the same set of offers for identical queries. To address these aspects, the Pilarcos trader arranges the service offers for each role randomly before starting the search. After a complete federation offer has been found, the offers are again ordered randomly and the search is begun anew. Possible duplicate federation offers are skipped. Since the federation offer space is typically quite large, this procedure works well to provide evenly distributed results.

Currently, all federation offers that match with the original incomplete federation offer are returned. It would be possible to add a *preference* expression, as in the CORBA Trading Service, according to which found federation offers would be sorted. Another possibility, more in line with the Pilarcos trading model, would be to add

Algorithm **Populate**(incomplete federation offer I):

1. Retrieve architecture definition and policy framework graphs of I from the type repository.
2. Collect candidate service offers for each role R_i as follows:
   (a) If there is a pre-filled offer in I for R_i, mark it as the only service offer for R_i; else
   (b) Search the service offer database for offers of R_i's service type.
3. Sort the roles $(R_1 \ldots R_n)$ into ascending order according to the number of candidate service offers.
4. Initialise empty list of complete federation offers L.
5. Invoke **SearchRole**(R_1, empty federation offer, set of empty graph-specific policy frameworks).
6. Return federation offer list L.

Algorithm **SearchRole**(role R_i, federation offer F, set of graph-specific policy frameworks P):

1. For all candidate service offers O_j of role R_i do:
   (a) Query type repository for compatibility of O_j's interfaces with adjacent offers in F.
   (b) If an interface is not compatible, return from the algorithm.
   (c) Calculate intersections of corresponding policies in P and O_j, storing the results back in P.
   (d) If any intersection is empty, return from the algorithm.
   (e) Add service offer O_j to federation offer F.
   (f) If R_i is the last role, then:
      i. Copy policy values from P to corresponding policy frameworks in the service offers of F.
      ii. Add F to federation offer list L.

      otherwise:
      i. Invoke **SearchRole**(role R_{i+1}, copy of F, copy of P).
2. Return from the algorithm.

Fig. 5. Outline of the Pilarcos trader population algorithm.

preferences to individual policies and sort the service offers according to them before starting the search.

The service offers traded include sets or intervals of alternative business policy values or technical property descriptions presented as service offer properties. These sets and intervals get narrowed down to values that are acceptable for the combination of services in the federation. However, the selection of final policy values or technical properties need to be unambiguous. For now, we let the Pilarcos trader choose those final values, without any further negotiation.

4 Cost of population process

The Pilarcos middleware feasibility is much dependent on the scalability of Pilarcos trading. The Pilarcos trader prototype implementation and measurement environ-

ments are introduced. The measurements show that the algorithm behaves well under changes in service offer space and business architecture complexity.

4.1 Pilarcos trader prototype implementation

The prototype implementations of the Pilarcos trader and type repository have been written in C++ on the MicoCCM [21] CORBA Component Model platform. They are independent parts of the larger Pilarcos prototype software. The Pilarcos trader implementation has two alternative modes of operation: standalone or using a standard CORBA Trading Service implementation for service offer storage. Standalone mode is considerably faster, since interprocess communication is avoided.

Using a CORBA Trading Service for service offer storage has a number of other advantages, however. Mature implementations are available, with different options for storing and caching service offers. Linking of CORBA traders is directly available, which enables the Pilarcos trader to scale to larger systems. By adhering to the standard, interoperability can be achieved, and it becomes possible to directly take advantage of the research efforts to improve the scalability of the CORBA trading model [22, 23].

If a CORBA Trading Service is used, on exporting the Pilarcos trader converts service offers into a form suitable for the CORBA trader and registers them there. The reverse conversion is performed during the population process. The Pilarcos trader imports service offers from the CORBA trader in blocks of five offers at a time to limit offer transfer overhead, since the Pilarcos trader and the CORBA trader reside in different processes and perhaps even on different servers.

In both operating modes, the Pilarcos trader prototype keeps the service offers in main memory. This is typical of traders, which need to have a small response time to queries. At the very least, the most frequently used service offers would be cached in main memory.

4.2 Measurement parameters

For performance measurements, computer-generated service offer databases and business architectures were used with separately controllable parameters. The measurement parameters:

Number of roles in business architecture. The generated business architectures consist of roles bound together in the form of a chain, without cross bindings; however, the form of the architecture makes no difference in the search algorithm. Each role in the architecture has a service type of its own.

Number of service offers. Service offers were generated separately for each role in the architecture. Their distribution was controlled by two parameters: minimum and maximum number of service offers per role. The number of offers for the roles was linearly interpolated between the two.

Number of policies. The total number of policies per service offer, distributed evenly between policy frameworks. One half of the policies were intervals and string set constraints, the other half was integers and boolean values.

Offer match ratio. This is the ratio of federation offers to all possible combinations of service offers in the offer database.

Since policy frameworks are attached to interfaces, the number of interfaces per role is not an independent parameter; adding interfaces has the same effect on performance as adding policies.

Per data point, 20 randomised service offer databases were generated. The cut-off role determining federation offer compatibility or incompatibility in the generated databases was the third role.

Parameters for the baseline case were: four roles, 40 to 100 service offers per three roles (210 total), four interfaces per role, 32 policies per service offer in four policy frameworks, and an offer match ratio of 30 %. This represents a rather heavy but, in our opinion, realistic usage scenario. In most tests, the Pilarcos trader was set to return 20 federation offers per request; however, in the results, only the time to find the first federation offer is reported, since it is largest and the differences are small. No time limit was set.

The measurement client created an incomplete federation contract with one pre-filled role, and called the Pilarcos trader `populate` operation to fill the remaining three roles. Thus, in the baseline case, the Pilarcos trader had a total of 280 000 possible service offer combinations to search, of which 84 000 were valid federation offers.

4.3 Measurement environment

The performance measurements were conducted on two 1 GHz Pentium III workstations with 512 MB of RAM, connected by a closed 100 Mbit Ethernet LAN. The measurement client program was run on one workstation and the Pilarcos trader on the other workstation. For the CORBA trader we used the Java-based ORBacus trader 2.0.0 [24] running on IBM Java2 1.4.0 in compiled mode on the same workstation as the Pilarcos trader.

Time spent in the Pilarcos trader search algorithm, time spent in the CORBA trader and the response time seen by the measurement client were measured separately. The Pilarcos trader and the ORBacus trader were restarted at the beginning of each measurement series, and were warmed up with three population requests before the actual measurements. For each data point, 20 runs were conducted, one per generated random database. The service offer database was emptied between each run.

4.4 Results and analysis

When a search limit of one federation offer was used, the population process in the baseline case took an average of 22 milliseconds. The Pilarcos trader was in standalone mode, as with other results except where noted otherwise. In addition, transferring the request and the result over CORBA took an average of 30 ms, raising the total to 52 ms. The federation offer data structure is designed for readability and flexibility, not speed: it contains several nested sequences and makes heavy use of the CORBA `Any` type, making marshalling very performance-intensive. The marshalling delay could be significantly reduced by using known CORBA IDL optimisation techniques. For the rest of the results, only the time spent in the search process is presented, since the marshalling delay is constant and predictable.

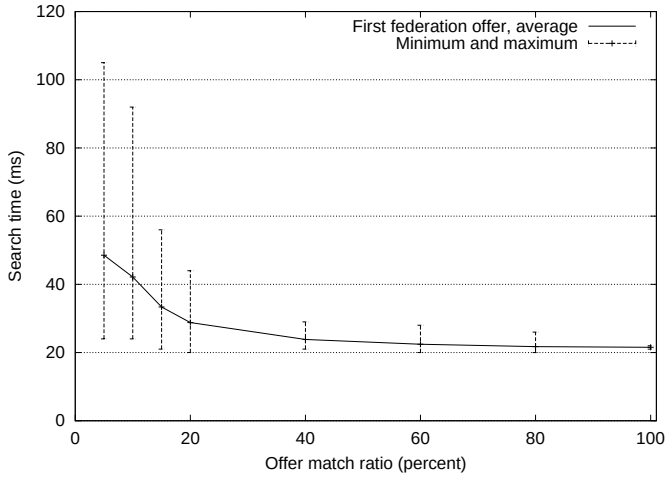

Fig. 6. Effect of offer match ratio on search time.

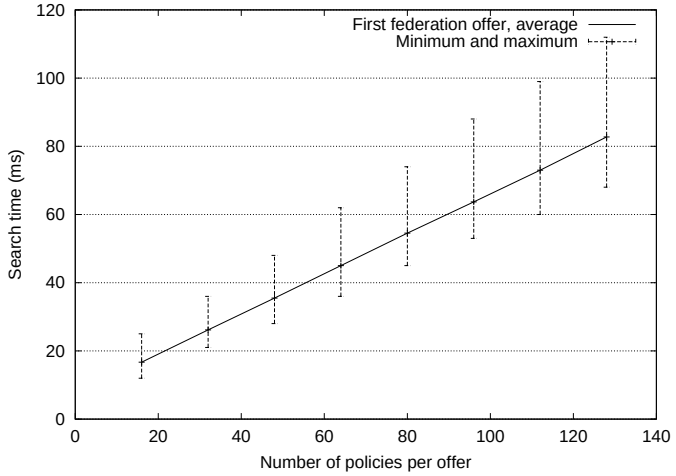

Fig. 7. Effect of number of policies on search time.

The result of varying offer match ratio from the baseline case is presented in Figure 6. Both the average time and the variation grow significantly with low match ratios, but are still tolerable even at a match ratio of 5 %. Based on these results, the practical usage area for the Pilarcos trader is with offer match ratios at and above 5 %.

Effect of the number of service offers was also measured. With offers distributed evenly between the roles, no significant effect on search time was seen with database sizes of up to 2550 offers. This behaviour was expected, since the population algo-

rithm never needs to search the entire offer database. Instead, search time is directly proportional to the number of requested federation offers.

In the measurement series presented in Figure 7, the number of policies per service offer was varied. As expected, search time grows in linear proportion to the number of policies. In the baseline case there were 32 policies per offer, which is a rather a high estimate, over 64 policies would be exceptional. In this respect, the Pilarcos trader scales very well. Moreover, the results could be improved significantly by reducing the amount of copying and insertion and extraction from CORBA `Any` types in the policy handling code.

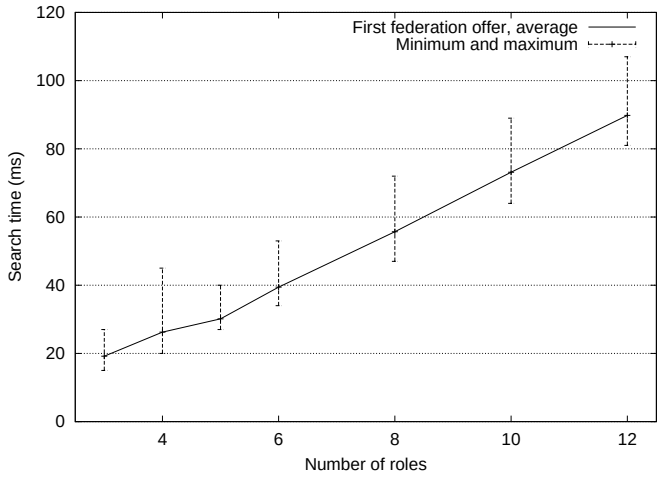

Fig. 8. Effect of number of roles on search time.

Figure 8 illustrates the effect of varying the number of roles in the business architecture. Again, the dependency is linear, as expected. Based on these results, large architectures with up to ten roles would be practical; most probably, for such large architectures other aspects than the population process are more significant.

For the measurements in Figure 9, ORBacus trader was used as service offer database. The search times are nearly ten-fold compared to the standalone case (Figure 6), with significantly larger variation. Also, an initial cost of over 100 ms is incurred by the first ORBacus trader queries. The initial cost as well as the large variation results from the block-wise transfer of service offers from the ORBacus trader. Additional, seemingly random fluctuations in the curve are probably caused by garbage collection in the Java process. According to more detailed measurements, transferring service offers between the ORBacus trader and the Pilarcos trader takes more than half of the ORBacus trader query time. When a CORBA trader is used as a back-end, a significant significant speedup could be achieved by collocating the Pilarcos trader and the CORBA trader in the same process.

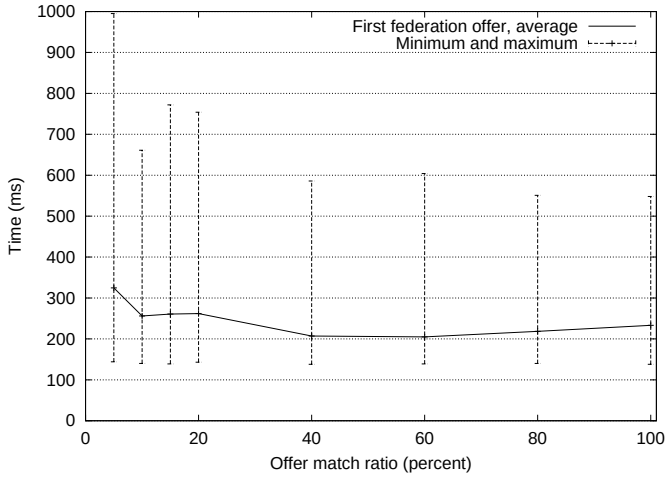

Fig. 9. Effect of offer match ratio on search time with ORBacus trader.

However, the fact that transferring service offers is so performance-intensive also has implications for using federated (linked) CORBA traders. In some federated cases, the service offers would be transferred across multiple links before reaching the Pilarcos trader, multiplying the performance costs. The CORBA Trading Service has not been designed for cases where the entire contents of service offers need to be transferred, and would need to be modified to support such cases practically. As with federation offers, this could be done with known CORBA IDL optimisation techniques, for example by replacing CORBA `Any` types by a more restricted set of possible property types.

5 Conclusion

The Pilarcos middleware is designed with service-oriented architectures in mind, capturing discovery of services and ensuring interoperability between them [5]. Currently, most service-oriented architectures built with Web services middleware, in Java environments or with CORBA support (plain or with components) concentrate on service discovery in client-server type of situations. Most middleware solutions have facilities for composing services into larger elements to be provided for inter-organisational use. Services are considered as a set of independent resources in the global network, as building blocks, from which an application programmer is responsible to construct a new added-value service. Most middleware solutions solve the interoperability problems by either trusting a shared language context (Java Virtual Machine) and transfer of code, or trusting a shared protocol environment (CORBA, Web services). In Pilarcos, interoperability tests use explicit information on the shared platform facilities, thus making it possible to use different technologies side-by-side as long individual federations find common communication facilities. The same goal appears to be set for OGSA (Open Grid Services Architecture) [25].

The Pilarcos model provides on multilateral contracts. A business architecture can be considered as an external workflow template that describes potential choreographies between independent parties. Each potential party provides a local workflow that fulfils its published service contract. The local workflows are hidden and no specific requirements for their running environment is specified. The Pilarcos middleware acts as a matchmaker for the parties, and then specialises the shared external workflow according to the membership, corresponding business policies and technical facilities. The design deliberately differs from the common goal for distributed workflow systems by not granting access and visibility to partner's information processing system.

The role of multilateral contracts composed of simple service descriptions and connections between them is parallel with Microsoft's XLANG specification [26]. The XLANG approach considers business process as a contract between two or more parties. An XLANG service description is used to define the behavior of each party. Moreover, the business process definition shows how the individual service descriptions are combined, using a map that defines the connections between the ports of the services involved.

Publishing new business process specifications is essential for the evolution of inter-organisational cooperation and service markets. In Pilarcos, business architecture descriptions are directly published in a repository and can thus be used as a structuring template for new federations. IBM's WSFL (Web Services Flow Language) approach defines a public interface that allows business processes to advertise themselves as Web services [27]. Both mechanisms have much the same effect. The recent BPEL (Business Process Execution Language) specification captures features of both WSFL and XLANG [28].

Providing federation management facilities in a middleware has the obvious benefit of releasing programmers from re-implementing these facilities repeatedly in applications. Furthermore, inter-organisational interoperation can only be achieved through standard solutions and in this case, via standardising new middleware services and metainformation elements.

Separating design of business architecture descriptions, design of service types, and implementation of components improves the software engineering process in general. The natural lifetimes of these elements differ, as well as the skills required for the provision of them. The Pilarcos middleware service allow workers to concentrate on their component logic and provides business architectures and technology mappings as ready-made solutions, as instructed by the currently popular production line concept [29].

Providing tools for automating administration of service composition reduces system maintenance cost. Because the federations directly support changes in membership, there is good support for adaptation to changes in business situation and also to changes in technical availability of services. The design incorporates into the same management model both private business within the organisation and external cooperations with varied and potentially contradictory requirements.

Future enhancements on the business architectures increase flexibility and adaptability of federations. We intend to expand business architectures with epochs and cardinalities of roles. An epoch is an interval during which services provided by a federation stay identical and the set of roles involved is unchanged. A change in services

or roles starts a new epoch. In addition, extensions are needed so that federations can be made to overlap or form hierarchies without application components to be involved in the administration.

The measurement results from the Pilarcos prototype show that the population process can be made scalable and the basic cost tolerable for the intended use. Pilarcos middleware is designed with dynamically changing cooperation schemes in mind. Such schemes appear for instance as virtual enterprises that provide a new service constructed by minor services run at member organisations computing facilities. Although there is a noticeable cost at establishing the federation, the duration of the federation is not only for single interactions but for a lengthier user session or more naturally, for the lifetime of the virtual enterprise.

The open nature of Pilarcos model is such that highly secure systems cannot be supported: trust building mechanisms can be incorporated, but still, critical systems should use more static and closed solutions. The nature of Pilarcos middleware overhead cost is such that it is not suitable for example for hard real-time systems or systems where response times are not allowed to vary a lot.

Future work moves Pilarcos from current component platforms to Web Services environment.

Acknowledgements

This article is based on work performed in the Pilarcos project at the Department of Computer Science at the University of Helsinki. The Pilarcos project is funded by the National Technology Agency TEKES in Finland, together with Nokia, SysOpen and Tellabs.

References

1. : Rosettanet implementation framework: Core specification v02.00.00 (2001) `http://www.rosettanet.org/`.
2. Austin, D., Barbir, A., Ferris, C., Garg, S.: Working draft on web services architecture requirements. Technical report, W3C (2002) `http://www.w3.org/TR/2002/WD-wsa-reqs-20020819`.
3. Wang, N., Schmidt, D., O'Ryan, C.: Overview of the CORBA Component Model. (2000)
4. Bodoff, S., etal: The J2EE Tutorial. (2002) http://java.sun.com/j2ee/tutorial/download.html.
5. Kutvonen, L.: Automated management of interorganisational applciations. In: EDOC2002. (2002)
6. Vhaho, M., Haataja, J.P., Metso, J., Suoranta, T., Kutvonen, L.: Pilarcos prototype II. Technical report, Department of Computer Science, University of Helsinki (2002) Draft, to be published in 2003.
7. Siegel, J.: Developing in OMG's Model-Driven Architecture. Object Management Group. (2001) White paper, revision 2.6.
8. ISO/IEC JTC1: Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 3: Architecture. (1996) IS10746-3.
9. ISO/IEC JTC1: (Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. ODP Type Repository Function) IS14746.

10. Lupu, E., Sloman, M.: A policy based role object model. In: Proceedings of the 1st International Enterprise Distributed Object Computing Conference (EDOC'97), Gold Coast, Queensland, Australia. (1997) 36–47
11. ISO/IEC JTC1: Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – ODP Interface References and Binding. (1998) IS14753.
12. Kutvonen, L.: Management of Application Federations. In Konig, H., Geihs, K., Preuss, T., eds.: International IFIP Working Conference on Distributed Applications and Interoperable Systems (DAIS'97), Cottbus, Germany, Chapmann & Hall (1997) 33 – 46
13. Object Management Group: OMG Trading Object Service Specification. (2000) OMG Document formal/2000-06-27. Also `http://www.omg.org/cgi-bin/doc?formal/2000-06-27`.
14. OASIS consortium: UDDI version 3.0 published specification. (2002) `http://www.uddi.org/specification.html`.
15. ISO/IEC JTC1: Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model for Open Distributed Processing. (1996) IS10746 1-4.
16. Kutvonen, L.: Trading services in open distributed environments. PhD thesis, Department of Computer Science, University of Helsinki (1998)
17. Vallecillo, A., Hernndez, J., Troya, J.M.: Component interoperability. Technical report, Dept. Lenguajes y Ciencias de la Computacin, University of Mlaga (2000) ITI-2000-37.
18. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. ACM Press and Addison-Wesley, New York, NY (1998)
19. Medvidovic, N., Taylor, R.N.: A framework for classifying and comparing architecture description languages. In Jazayeri, M., Schauer, H., eds.: ESEC/FSE '97. Volume 1301 of Lecture Notes in Computer Science., Springer / ACM Press (1997) 60–76
20. Allen, R.J.: A Formal Approach to Software Architecture. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh (1997)
21. : MicoCCM project web pages. (2002) `http://www.fpx.de/MicoCCM/`.
22. Craske, G., Tari, Z., Kumar, K.R.: DOK-trader: A CORBA persistent trader with query routing facilities. In: International Symposium on Distributed Objects and Applications. (1999) 230–240 Also `http://rmit.edu.au/~zahirt/Papers/doa99.pdf`.
23. Belaid, D., Provenzano, N., Taconet, C.: Dynamic management of CORBA trader federation. In: 4th USENIX Conference of Object-Oriented Technologies and Systems (COOTS), Santa Fe, New Mexico (1998) Also `http://www.usenix.org/publications/library/proceedings/coots98/full_pap%ers/belaid/belaid.pdf`.
24. IONA Technologies: ORBacus Trader, version 2.0.0. (2001) `http://www.iona.com/products/orbacus/trader.htm`.
25. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: Grid services for distributed system integration. IEEE Computer **35** (2002) 37—46
26. Thatte, S.: Xlang. Technical report (2001) `http://www.gotdotnet.com/team/xml\_wsspecs/xlang-c/default.html`.
27. Leymann, F.: Web services flow language (wsfl 1.0). Technical report (2001) `http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`.
28. Thatte, S.: Business process execution language for web services, version 1.0. Technical report (2002) `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/`.
29. Herzum, P., Simms, O.: Business Component Factory. Wiley Computer Publishing (1999)