An architecture for certification-aware service discovery

Michele Bezzi*, Antonino Sabetta* and George Spanoudakis†
*SAP Research Sophia-Antipolis
805, Av. du Docteur Maurice Donat - 06250 - Mougins, France
Email: {firstname.secondname}@sap.com
† School of Informatics, City University London
Northampton Square, London, EC1V 0HB, UK.
E-mail: g.spanoudakis@soi.city.ac.uk.

***Abstract*—Service-orientation is an emerging paradigm for building complex systems based on loosely coupled components, deployed and consumed over the network. Despite the original intent of the paradigm, its current instantiations are limited to a single trust domain (e.g., a single organization). Also, some of the key promises of service-orientation - such as the dynamic orchestration of externally provided software services, using run-time service discovery and deployment - are still unachieved. One of the main reasons for this is the trust gap that normally arises when software services, offered by previously unknown providers, are to be selected at run-time, without any human intervention.**

To close this gap, the concept of machine-readable security certificates (called *assert*s) has been recently introduced, which paves the way to automated processing about security properties of services. Similarly to current security certification schemes, the assessment of the security properties of a service is delegated to an independent third party (certification authority), who issues a corresponding assert, bound to the service. In this paper, we propose an architecture, which exploits the *assert* concept to realise a certification-aware service discovery framework. The architecture supports the discovery of single services based on certified security properties (in additional to the usual functional properties), as well as the dynamic synthesis of service compositions, that satisfy the given security properties. The architecture is extensible, thus allowing for a range of domain-specific matchmaking components, to cover dimensions related to, e.g., performance, cost and other non-functional characteristics.

I. Introduction

Service-based software systems engineering has emerged as a paradigm for building complex software systems based on loosely coupled components, known as software services, that can be deployed programmatically over networks. Supported by industry standards for the description of software service interfaces (WSDL [8]), message based service communication mechanisms (SOAP [19]), the new paradigm has gained ground as a way of achieving interoperability. The uptake of this new paradigm, however, has not been extensive in the case of software assets that cross organisational boundaries and are under the control of autonomous parties (service providers) and as of today is only adopted within highly controlled environments (e.g., within the IT infrastructures of single organisations).

One of the main reasons for this limitation is the trust deficit that normally arises in such circumstances. In particular, users are concerned about the security of external software services, which are not under the control of the system provider (integrator), and about the communications between their service-based system and these services. Unfortunately, this lack of centralised control is a typical characteristic of cloud-based applications.

This security deficit becomes even more significant in cases where a service-based system needs to consume new services that are dynamically selected at run-time. This may happen due to the unavailability of the services that were bound to the system originally or to their failure to satisfy given system requirements (e.g., performance). In such cases, beyond the need to ensure that any new candidate service satisfies the interface and behavioural conditions, which is a prerequisite for using it as part of the system, it is also necessary to ensure that the service satisfies the *security requirements* associated with its deployment.

Several techniques have been developed to enable the dynamic adaptation of service-based systems (SBS) for the deployment of new services at run-time. These techniques vary in sophistication, ranging from those supporting the discovery of single services (e.g., [16], [18], [22], [29]) to more complex counterparts aiming to creating orchestrations of more than one services that can replace the one that has failed [28], [7], [6]. Regardless of their overall aim and outputs, however, existing techniques fail to provide comprehensive support for reasoning about security properties at run-time and providing the assurance required due the deployment of new services. This is due to the fact that existing techniques either do not take security conditions at all into account or they focus only on the security of the communication between the service-based systems (SBS) and the externally-provided service, without considering security properties of the internal behaviour of the latter (e.g., whether and how it stores confidential data, whether it communicates confidential data to third parties, whether it allows unauthorised modifications to data, and so on). Furthermore, current mechanisms for the dynamic creation of service orchestrations, as replacements of single services within a system, fail to ensure that the resulting orchestration will preserve the security properties of the original service.

In this paper we propose the architecture of a novel service discovery platform that aims to bridge these gaps. The key

characteristic of this platform, which is being developed in the context of the ASSERT4SOA[1] project, is the use of machine-readable, signed statements, called *asserts* [21], that certify the security properties owned by a service. Similarly to current security certification schemes, the assessment of the security properties of a service is delegated to an independent third party (certification authority), who issues a corresponding assert, bound to the service. The certification of a security property in an assert is based on either a formal proof or on service testing that has been carried out before the certificate is issued. These formal proofs and tests must have been carried by the entity who has issued the certificate (e.g., a certification authority). Differently to existing security certificates (e.g., Common Criteria [13]), asserts are represented in a form that is suitable for automated reasoning and processing. Asserts include not only the specification of the security property that they guarantee for the service but also information about the certification authority that has issued the certificate and a description of the evidence that underpins it. Hence, the certificates assumed by our approach provide comprehensive descriptions of security properties that can be queried and analysed at run-time by the discovery platform without a need for human intervention. As a result, they can be successfully employed in a service discovery platform; based on the information encoded into asserts, it is possible to query for services that satisfy certain security properties, filtering out those that do not. Also, by reasoning on the content of asserts, such a discovery system can rank the matching candidate services in a way that enables automatic service selection at run-time.

The service discovery platform proposed in this paper is also characterised as follows.

- Its operation is driven by queries specified by SBS designers during the system development process. These queries are executed at run-time to discover services without any further need for human intervention.
- In addition to security properties, it provides support for assessing interface, functional, quality properties during run-time service discovery.
- It supports the discovery of single services as well as service compositions which are synthesised at run-time based on a given set of composition patterns which are proven to satisfy given security properties if the individual services composed by them have certain certified security properties themselves.
- It advocates an extensible architecture where, in addition to a built-in set of core matchmaking components for assessing the compatibility of service interface and security properties, the platform can be extended with domain-specific matchmaking components for assessing service quality, behavioural properties, non-functional properties other than security. Special matchmaking components may be used for analysing and reasoning about the evidence (proofs, tests) carried by asserts, in cases where this is necessary for additional assurance in specific domains or circumstances (e.g., when a specific type of assessment is required).
- It supports the execution of service discovery queries in both reactive and proactive modes, i.e., following a request by the SBS that acts as a client to the platform to replace a specific service (reactive mode) or continually and in parallel with the operation of the SBS in order to ensure that when the constituent service of the system for which the query has been constructed becomes unavailable or fails to satisfy the needs of system, e.g., the terms established in a Service Level Agreement (SLA), possible replacement services for it will have been already identified (proactive mode).

The assert-aware service discovery platform, proposed in this paper, is being developed as an extension of the SERDIQUEL run-time service discovery system described in [28], [29]. Hence, the main contribution of this paper is to integrate and extend the original SERDIQUEL architecture to provide the features listed above.

The remainder of the paper is structured as follows. In Section II, we present a sample motivating scenario highlighting the shortcomings of discovery as it is commonly exposed by today's systems and suggesting a potential application of our proposal. In Section III, we describe the high-level architecture of the service discovery platform and the role of its main components. Section IV details how these components interact to realize the discovery and matchmaking functionality. In Section V we provide a review of related literature and highlight our contribution with respect to it. Finally, in Section VI, we draw some conclusions and outline directions for future research and developments.

II. MOTIVATING EXAMPLE

SecureMail Inc.[2] offers customizable e-mail services over a cloud-based infrastructure. Customers use SecureMail services by accessing a SBS that offers basic e-mail messaging functionality; such an SBS can be customised by adding one or more complementary add-ons, provided over the cloud, offered and run as services by third parties, in order to have a messaging service tailored the specific needs of each customer. As an example, users may pick an antivirus add-on for detecting malicious attachments, a spam-filter to reduce unwanted mails, an add-on to support digital signing of messages, and a backup function to archive old messages in a reliable, searchable, and secure way.

SecureMail offers to users an interface whereby they can specify what added functionality they need, as well as the security preferences referred to each of these functionalities. For example, a customer may indicate that only antivirus products that are certified by a particular certification agency should be used. Additionally, both the anti-virus and the anti-spam functions should only be delegated to third-party providers that are certified as being capable of communicating with SecureMail's servers through an strongly encrypted channel. Also, they must ensure that the privacy of the customer is

[1] http://www.assert4soa.eu

[2] This is an imaginary company name, invented for the purposes of this paper.

preserved by disallowing any use of the content of his/her e-mails. Finally, the backup of messages should be performed in such a way that the backup is encrypted on SecureMail's server *before* being sent over the network to the storage server, in order to ensure that the contents of the messages is kept confidential. The price, availability and overall performance of each individual add-on service may be different, and the customer can specify his/her preferences also about these aspects.

Based on all these preferences (functional, security-related, and other extra-functional constraints) the e-mail service offered by SecureMail discovers and presents to the user only the best available add-on service for each of the additional functions chosen by the customer, by taking into account his/her security preferences. Also, based on these preferences, it can act as a transparent delegate agent for the customer, by selecting and connecting dynamically to alternate services in case a better match if found during run-time (e.g., an alternative anti-spam service, with higher privacy guarantees, or one with different but equivalent guarantees, but lower price or better performance).

III. Architecture Overview

The service offered by imaginary company SecureMail, as described in the previous section, relies heavily on the existence of an underlying sophisticated discovery platform. Such platform not only is capable of understanding and processing discovery requests involving a complex set of dimensions, but also to be responsive to changes in the landscape of available services by providing fresh, up-to-date results for a given query.

This section and the following are dedicated to presenting a high-level architectural view of a discovery system that is capable to address scenarios such as the SecureMail one. In particular, we describe the key functionalities that the architecture is meant to provide and how these functionalities are allocated to the fundamental components. The next section concentrates on describing the mechanisms underlying the matchmaking functionality. Throughout this and the following section, we refer to the component diagram depicted in Figure 1.

A. Key functions

In a nutshell, the architecture presented in this paper is meant to provide a comprehensive framework through which the full life-cycle of services with asserts can be managed, spanning assert issuing and management, assert-aware discovery and matchmaking, assert verification, and finally service consumption. At a very abstract level, the components that make up the architecture can be partitioned into a few coarse-grained functional areas, as outlined below.

Assert issuing and management. These components (upper-left dashed box in Fig. 1 and Registration Manager in the bigger dashed box) include the tools and user interfaces that allow assert issuers and managers to create asserts and to manage their life-cycle (i.e., their issuing, update, and revocation).

Query handling and notification management. Queries coming from consumers are interpreted and then used as the basis for governing the overall discovery and matchmaking process. As part of the query handling, the system provides different interrogation mechanisms. More precisely, besides the basic request-response interaction, a publish-subscribe interrogation style is supported. In the latter case, the query handling function is also responsible for creating a response channel corresponding to each query, whereby results are pushed to the interested consumers.

Service discovery and matchmaking. This functionality represents the core of the system. Its overall functioning is ensured by a Discovery Manager, which acts as a coordinator of several different subsystems, including the matchmaking subsystem, the composition manager, as well as the registry abstraction layer.

Consumer-side and boundary elements. On the consumer side (upper-right dashed box in Fig. 1) a consumer agent and a few additional support elements are deployed. The former is the the component through which the consumer (typically a SBA) can interact with the server-side elements of the system. The support components are dedicated to tasks such as the consumer-side verification of asserts, the management of trust relationships with third-party service providers, and the handling and storage of consumer preferences concerning security properties. Boundary elements include the server-side components that are concerned with the interaction with the consumer (Frontend) and with back-end service registries (Registry Abstraction Layer).

System management and auditing. The architecture provides management and configuration functionalities, as well as security-related functions so as to guarantee that the ASSERT4SOA framework itself is secure (e.g., by ensuring that only authorised users can access certain functionalities, and that any relevant events of the framework are captured in a secure audit trail).

B. Main components and subsystems

The macro-functionalities outlined above, are detailed in following, describing the responsibilities of the individual components. For the sake of conciseness, some inessential components are left out of this description. Also, in order not to clutter the diagram in Figure 1, some interfaces have been represented in an abstract way as dependencies, and some dependencies have been omitted (e.g., between most components and the Audit System).

Frontend. This is the common entry-point providing a uniform API through with consumer-side and issuer-side components can interact with the server-side elements of the discovery system. The Frontend provides access control functionalities and provides an important source of security relevant events, that are captured in a secure Audit Trail. Additionally, the Fronted may offer an interface through which services can be registered in back-end service registries (although this functionality is optional, as registration may as well occur directly at one of the back-end service registries).

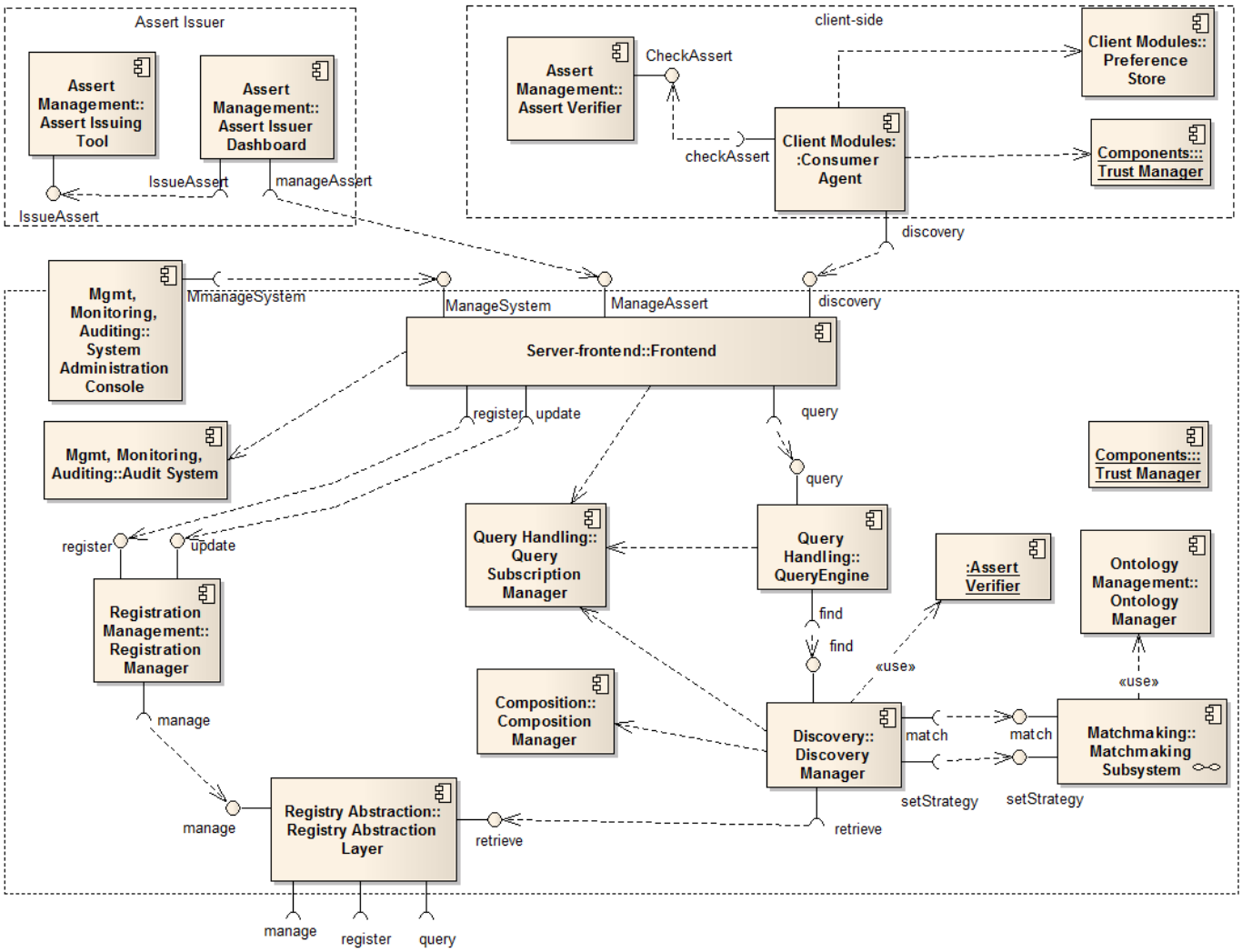

Fig. 1. Architecture Overview

Query Engine. This component is connected to the Frontend on one side and with the Discovery Manager on the other. It is in charge for parsing the query coming from the consumer and passing it on to the Discovery Manager. Queries contain both a specification of the functionality that candidate services must offer as well as a set of conditions on their non-functional properties. The primary concern addressed by this component is to decouple the query language used by the service consumer from the language that is used internally by the ASSERT4SOA framework, and especially by the Discovery Manager.

Query Subscription Manager. The queries coming from the client may specify a subscription query; in this case, the query is continuously evaluated and the client is notified when new matching services are found. The Query Engine interacts with the Query Subscription Manager in order to create and manage query-specific response channels, whereby a consumer can subscribe to a query and can receive continuous updates as new matches are found that satisfy that query.

Discovery Manager. Based on the information coming from client-supplied complex queries, as processed by the Query Engine, the Discovery Manager coordinates several different subsystems. Firstly, it uses the Registry Abstraction Layer to retrieve an initial set of candidate services, based on their functional description. Secondly, it instantiates a matchmaking strategy. Such a strategy is basically a description of how various *types* of matchmaking modules should be coordinated and how their results should be aggregated in order to determine the final list of candidate services, ranked according to their degree of fit, to be returned to the consumer. The strategy and the initial list of services obtained from the Registry Abstraction Layer represent the other input to the Matchmaking Subsystem. Finally, the Discovery Manager activates the Composition Manager and coordinates its operation as explained at the end of the next section.

Matchmaking Subsystem. This subsystem is controlled by the Discovery Manager which activates it by passing as input (a) an initial set of (functionally matching) candidates, and (b) a matchmaking strategy (produced based on the contents of the query). As a result of the matchmaking process, the

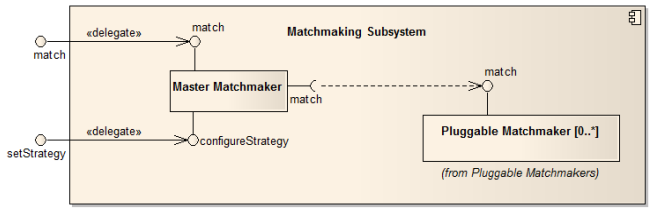

Fig. 2. Matchmaking subsystem

candidate services are filtered (discarding those that do not match the non-functional requirements) and ranked according to their degree of fit. Internally, the Matchmaking Subsystem is organized as a hierarchical, dynamically configurable architecture. It is *hierarchical* since a Master Matchmaker controls a set of Slave Matchmakers and aggregates in a single fit measure the results coming from each slave; it is dynamically configurable since the organisation of the slaves is determined and realised at run-time, based on the matchmaking strategy, which in turn is determined based on the query.

This design allows each slave to be realised as very targeted, domain-specific evaluator of a particular property or dimension, whereas the master matchmaker is only concerned with the coordination of slaves. In this way, additional (or alternative) slave matchmakers can be plugged into the system, thus supporting the evaluation of an extensible range of properties. While the focus of ASSERT4SOA is on security-related properties, the architecture accommodates a sophisticated coordination of different pluggable matchmaking components, in such a way that the decision as to which candidate is to be chosen can be taken on a more comprehensive basis (e.g., capturing constraints related to performance, availability, cost, and so on). Slave matchmakers may be provided by external third-party services, allowing for an additional level of dynamism and diversity, possibly enhancing availability and fault-tolerance (although raising, at the same time, additional security and trust concerns).

Assert Issuing Tool. This component allows assert issuers to express the results of their assessment as an assert. The tool provides a graphical user interface that guides the user (typically a certification authority) in the process, and produces as output an assert that conforms to the Assert XML-schema and that is digitally signed by the issuer.

Assert Verifier. The Assert Verification Module is responsible for checking that an assert is valid. This component is used both server-side, before the results of matchmaking are pushed to the consumer, and client-side, where the consumer may want to check on his/her own the asserts, before consuming a service. The verification involves several steps. Firstly, the signature on the assert is checked to ensure the assert is authentic. Secondly, the well-formedness of the assert is verified. Finally, the credentials of the assert issuer are checked, based on the preferences of the client (assert consumer).

It is worth noting that, in principle, any entity with a digital identity can play the role of assert issuer, since an assert is nothing but a signed statement on the properties of the subject service. The problem of establishing the trustworthiness of such a statement, which is fundamental to use asserts in practice, may be addressed in different ways: for example, the verification module could verify that the issuer is in the list of trusted authorities, or that the issuer does have an accreditation received from a suitable accreditation body. Although it may seem extreme to allow anybody to play the role of an assert issuer (i.e., of a certifier), this is very much like the approach currently accepted for identity certificates. As of today, anybody can sign identity certificates, where he/she claims that a certain public key corresponds to a given "identifier". And in fact, the decision as to who is trusted for assigning such certificates is left to the consumers, who typically express their decision through their web-browser's configuration options. This approach, however, may not be adequate for service-based applications. Such applications integrate software services of third parties in order to provide value-adding services to their own users/customers. Hence, the providers of SBAs have responsibility to undertake certificates checks, in automated ways whilst their systems are in operation and in a manner that reduces the risks for the users of their systems and, therefore, themselves.

Registration Manager. This component is responsible for publishing certified services on the registries available through the registry abstraction module and for managing their lifecycle (e.g., to update or revoke them). The registration module communicates both with the monitoring and with the auditing module (to log both functionally- and security-relevant events) and the authorization module.

Registry Abstraction Layer. The input request as coming from the Query Engine is split by the Discovery Manager into two parts that are treated separately: 1) the characterization of security properties required by the client, and 2) the description of the interface, functionality and other non-security-sensitive QoS conditions that are expressed as part of service discovery queries. Based on the latter, the Discovery Manager queries in turn the Registry Abstraction Layer to retrieve a set of candidate services that satisfy the required interface, functional and non-security characteristics required of services. While service description may be stored in several (possibly heterogeneous) back-end registries, the Registry Abstraction Layer provides uniform access to such back-end registries, regardless of the differences in their interfaces and protocols.

IV. DISCOVERY AND MATCHMAKING

In the previous section we introduced, among the others, the architectural elements related to the *discovery and matchmaking* functionality. In this section, we concentrate on describing how these components *interact*. The principle of discovery and matchmaking is illustrated as a sequence diagram in Figure 3. For the sake of conciseness, we omit the Frontend from this model, and we start the description from the point where the Query Handler has already been activated by receiving a query, forwarded by the Frontend.

A query, expressed using SERDIQUEL, contains a precise specification on the characteristics that must be possessed by candidate services in order to match the needs of the consumer. This characterisation covers functional as well as non-functional (in particular, security) attributes.

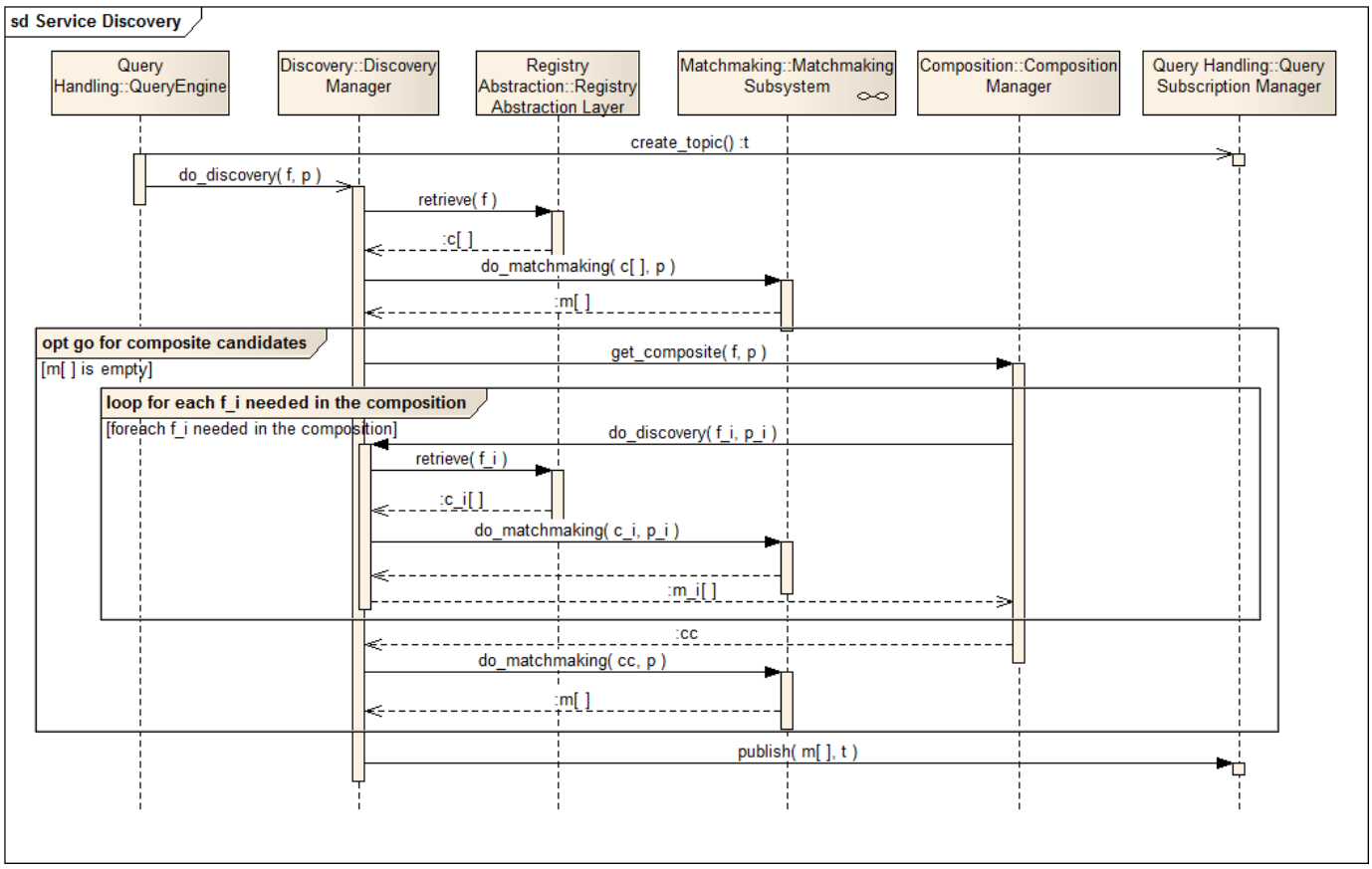

Fig. 3. Discovery subsystem

As a first step, the Query Engine activates the Query Subscription Manager, which is responsible for creating a response channel and a corresponding buffer through which the results of the query are made available to the client. This mechanism allows for a pub-sub interaction pattern. In this way, a client can register for a query and *stay tuned* on the corresponding response channel, so that it can receive fresh results as soon as they are produced. Especially, a response channel of this sort can be seen as a constantly updated pool of replacement services that match a given query. Also, the Query Engine takes care of parsing the query in such a way that it is readily consumable by the Discovery Manager and the other components that must handle it. In particular, the information conveyed in the query is decomposed so that the characteristics related to functional aspects are separated from those related to non-functional aspects. Also, based on the contents of the query, each piece of information is augmented with a specification of the confidentiality level with which it has to be processed. This allows to determine whether or not each part of the query can or cannot be forwarded to elements of the architecture that are provided externally by third parties.

The next step is the activation of the Discovery Manager, which, as we explain in the following, is the central coordinating element for the realization of the discovery and matchmaking functionality.

The Discovery Manager forwards the information describing the functional requirements to the Registry Abstraction Layer. This information will be effectively handed to the registries attached to this layer in accordance with the trust relationship that exists between the client that originated the query and each of the registries. As a result of this interaction, a set of candidates matching the functional specification expressed in the query is returned to the Discovery Manager.

This set of candidates, together with a matchmaking strategy, is then passed to the Matchmaking Subsystem, which operates on it as described in the previous section. The result of this step can be seen as a filtering of the candidates that discards those that do not meet the security assurance required by the client and that sort those that do match, according to their degree of fit. The resulting (filtered and ranked) set is then returned to the Discovery Manager. Finally, the Discovery Manager pushes these results to the corresponding response channel. A new discovery and matchmaking can be activated autonomously by the ASSERT4SOA system on a periodic basis, according to the request coming from the client and expressed in the query. Any new results are sent to the response channel; analogously, those services that did match the query previously but that no longer do, are removed from the channel. This may happen, for example, if the assert associated to a given service has changed, or if a service has become unavailable in the meantime.

Composition. An important feature of the ASSERT4SOA framework is its ability to synthesise, on-the-fly, service compositions that match a given query. This capability is activated, in particular, to cope with the case where no single service can be found that matches the query (because either the

functional or the security characteristics of the candidates do not match with the query). In this case, the outer optional fragment in the sequence diagram of Figure 3 is executed. The Composition Manager explores the space of possible compositions by applying *patterns* taken from a library of pre-defined composition patterns. Each pattern specifies the preconditions that each of the constituent service must fulfil as well as the security properties that are guaranteed to hold for the composite service, based on the assumption that the preconditions hold. Of course a large number of possibilities for composition is discarded, but dealing with arbitrary dynamic compositions, and especially, determining the properties possessed by such arbitrary compositions, would be a prohibitively hard problem to address. When a suitable composition is identified, it is still formulated in terms of abstract service placeholders. Each of these placeholders need to be filled in with a concrete service, whose functional and security characteristics correspond to the requirements of the composition. This means that for each service placeholder, a new discovery request is originated and sent to the Discovery Manager. Each such request is processed as described above, entailing a retrieval step from the Registry Abstraction Layer and then a matchmaking step. Of course, recursive activations of the composition mechanism are possible, although the allowed levels of nesting of such requests must be kept to a minimum, as it may have a considerable impact on the utilization of computing resource and on response time (parameters determining how composition patterns should be applied are specified as part of the discovery queries). When a matching composition is found, it can be returned either as an abstract composition specification, or as an runnable composition. In the former case a description of the service composition, along with the endpoints of suitable constituent services, is returned to the consumer, who is then responsible for running the composed service on his/her side. In the latter case, the discovery framework itself uses its built-in service orchestrator to realise the composition and to return an endpoint to the consumer. In this case, the composite service is consumed as any other (single) service, but it does not have a real assert, only a dynamically synthesised surrogate (which we call a *virtual assert*), derived using the composition rules.

V. Related work

A key capability in service-based applications is the ability to discover services that fulfil given functional and quality properties in automated ways at run-time. This capability is necessary in order to be able to replace existing services of a service-based application that might become unavailable or fail to fulfil the properties required from them due to a variety of reasons (e.g., context changes, variations in the deployment conditions of a service etc.).

Run-time service discovery has been the focus of several strands of work, including approaches supporting the discovery based on service interface, behavioural, and quality properties as well as combinations of them [12], [18], [25], [22]. Existing approaches support run-time service discovery based on the use of information retrieval techniques (e.g., [1], [2], [16]), complex graph matching algorithms (e.g., [15], [12], [28], [29]), or reasoning based on semantic descriptions of services [3], [15], [16], [17]. Considerable effort has also been concentrated on support for context awareness in service discovery, i.e., the ability to trigger the discovery process and modify the criteria used in it in response to changes in the deployment context of a service-based application and/or the services deployed by it [5], [9], [11], [14], [20], [25].

Existing approaches to service discovery can also be distinguished into those supporting the discovery of single services (e.g., [16], [18], [22], [29]) and those supporting the discovery of compositions of services [28], [6], [7]. The latter approaches are aimed to cover cases where no single service satisfying a given request can be located. Despite the existence of several approaches to run-time service discovery, only few of them take into account security properties [7], [6], [4], [23] and most of them concentrate on the discovery of services in ad-hoc networks [10], [26], [27], [24]. Most of these approaches focus on securing the discovery process itself [23], [10], [26], [27], [24] particularly in connection with the discovery of network services in pervasive environments (e.g., secure publication of service descriptions, secure requests for service discovery) rather than being concerned with discovery of services based on required security properties of these services. Other approaches focus on the security properties of the services to be discovered but tend to support only specific kinds properties (e.g., properties related to the secure interaction with a service, service support for particular authentication and authorization mechanisms [7], [6], [4]). Hence existing approaches fail to provide comprehensive support for more complex security properties (e.g., confidentiality and privacy preserving storage of information, internal transactional integrity). Furthermore, there is limited support for ensuring security in service compositions. Some of the work that focuses on composition (e.g., [7], [6]) is limited, as it does not support the propagation of the need to satisfy security properties within the constituent services in a composition.

VI. Conclusions

In this paper we presented an architecture for an advanced service discovery system that is capable of processing queries including constraints on the security assurance level provided by services. This is made possible by expressing the security properties of services in a format, defined in the context of the ASSERT4SOA project, that enables automated processing of security certificates.

The ambitious vision of the ASSERT4SOA project aims at overcoming the shortcomings of security certification as it is meant today, by introducing automated reasoning about certified security properties, in particular w.r.t. service discovery and matchmaking. The realisation of such vision will necessarily rely on the development of a reference prototype implementation of the architecture described in this paper, which we plan to undertake in the coming months.

The contribution presented in this work presents our initial results in this direction. It identifies and documents our high-level architectural design choices, listing the key function-

alities that the ASSERT4SOA framework will provide, and presenting the architecture in terms of a set of high-level components as well as the key interactions among them.

At this time, several problems remain open, and they require to be addressed before ASSERT4SOA can be turned into a viable technology.

An important fundamental problem to address is the *binding* problem. When dealing with the certification of services (as opposed to software products that are shipped in binary or source format to the consumers), the binding between an assert and the certified service is not trivial. In general, *without further assumptions*, it is not possible for the consumer to be sure that the service assessed by the assert issuer (i.e., the actual service realisation that the assert refers to) and the one offered at a given point in time by a service provider are actually the same piece of software. Exploring which are the minimal additional assumptions to make in order to ensure such a binding will be the topic of our future research.

Another challenging issue to address was somewhat hinted at the end of Section IV. The system we have designed is meant to be capable of synthesising composite services, based on composition patterns that are known to preserve (or assure) certain security properties. As we mentioned earlier in the paper, the guarantees provided by such a composite service cannot be supported by a proper assert (signed off-line by an assert issuer). At the same time, probably it is not realistic to assume that the entity running the ASSERT4SOA system be ready to take the responsibility of endorsing a dynamically synthesised *virtual* assert, because of the legal implications and liabilities that this would entail. More in general, our research agenda includes a more detailed analysis of possible realistic scenarios and an in-depth reflection about the use-cases where composition is involved, also with regard to the trade-offs between accuracy of results (achieved, e.g., by allowing multiple levels of recursion in the composition) and performance.

ACKNOWLEDGEMENTS AND DISCLAIMER

This work was partly supported by the EU-funded project ASSERT4SOA (grant no. 257361). Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of their institutions or of the funding agency.

REFERENCES

[1] Seekda. http://webservices.seekda.com/.

[2] Service finder. http://demo.service-finder.eu/search.

[3] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint Driven Web Service Composition in METEOR-S. *IEEE International Conference onServices Computing 2004 SCC 2004 Proceedings 2004*, pages 23–30, 2004.

[4] M. Bartoletti, P. Degano, and G. Ferrari. Enforcing secure service composition. In *Computer Security Foundations, 2005. CSFW-18 2005. 18th IEEE Workshop*, pages 211 – 223, june 2005.

[5] F. Bormann, S. Flake, J. Tacken, and C. Zoth. Towards Context-Aware Service Discovery: A Case Study for a new Advice of Charge Service. *14th IST Mobile and Wireless Communications Summit*, 2005.

[6] B. Carminati, E. Ferrari, R. Bishop, and P. Hung. Security Conscious Web Service Composition with Semantic Web Support. In *IEEE 23rd International Conference on Data Engineering Workshop*, number 60473091, pages 695–704. Ieee, 2007.

[7] B. Carminati, E. Ferrari, and P. K Hung. Security Conscious Web Service Composition. In *2006 IEEE International Conference on Web Services ICWS06*, volume 0, pages 489–496. Ieee, 2006.

[8] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawaran. Web services description language (wsdl)version 2.0 part 1: Core language. http://www.w3.org/TR/wsdl20, 2006.

[9] S. Cuddy, M. Katchabaw, and H. Lutfiyya. Context-aware service selection based on dynamic and static service attributes. *WiMob2005 IEEE International Conference on Wireless And Mobile Computing Networking And Communications 2005*, 9:13–20, 2005.

[10] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An Architecture for a Secure Service Discovery Service. In *Proceedings of the 5th Annual ACMIEEE International Conference on Mobile Computing and Networking MobiCom99*, pages 24–35, 1999.

[11] C. Doulkeridis, N. Loutas, and M. Vazirgiannis. A System Architecture for Context-Aware Service Discovery. *Electronic Notes in Theoretical Computer Science*, 146(1):101–116, 2006.

[12] D. Grigori, J. C. Corrales, and M. Bouzeghoub. Behavioral matchmaking for service retrieval. In *ICWS*, pages 145–152. IEEE Computer Society, 2006.

[13] ITSEC. Common criteria for information technology security evaluation.

[14] A. Karmouch and M. Khedr. Enhancing Service Discovery with Context Information. *ITS 2002*, 2002.

[15] U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic Location of Services. pages 38–49. Springer, 2005.

[16] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems AAMAS 06*, page 915, 2006.

[17] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 331–339, New York, NY, USA, 2003. ACM.

[18] R. Mikhaiel and E. Stroulia. Examining usage protocols for service discovery. In A. Dan and W. Lamersdorf, editors, *ICSOC*, volume 4294 of *Lecture Notes in Computer Science*, pages 496–502. Springer, 2006.

[19] H. F. Nielsen, N. Mendelsohn, J. J. Moreau, M. Gudgin, and M. Hadley. SOAP version 1.2 part 1: Messaging framework, June 2003.

[20] P. Pawar and A. Tokmakoff. Ontology-based context-aware service discovery for pervasive environments, 2006.

[21] J.-C. Pazzaglia, V. Lotz, V. C. Cerda, E. Damiani, C. Ardagna, S. Grgens, A. Maa, C. Pandolfo, G. Spanoudakis, F. Guida, and R. Menicocci. Advanced security service certificate for soa : Certified services go digital ! *Securing Electronic Business Processes*, Information Security Solutions Europe, 2010.

[22] Z. Shen and J. Su. Web service discovery based on behavior signatures. In *Services Computing, 2005 IEEE International Conference on*, volume 1, pages 279 – 286 vol.1, july 2005.

[23] S. Trabelsi, J.-C. Pazzaglia, and Y. Roudier. Secure Web Service Discovery: Overcoming Challenges of Ubiquitous Computing. In *Proceedings of the European Conference on Web Services*, pages 35–43. IEEE Computer Society, 2006.

[24] J. Undercoffer, F. Perich, A. Cedilnik, L. Kagal, and A. Joshi. A Secure Infrastructure for Service Discovery and Access in Pervasive Computing. *Centaurus*, 8(2):113–125, 2003.

[25] Y. Ye and G. Fischer. Context-aware browsing of large component repositories. In *Proceedings of the 16th IEEE international conference on Automated software engineering*, ASE '01, pages 99–, Washington, DC, USA, 2001. IEEE Computer Society.

[26] F. Zhu, M. Mutka, and L. Ni. Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, PERCOM '03, pages 235–, Washington, DC, USA, 2003. IEEE Computer Society.

[27] F. Zhu, M. W. Mutka, and L. M. Ni. A Private, Secure, and User-Centric Information Exposure Model for Service Discovery Protocols. *IEEE Transactions on Mobile Computing*, 5:418–429, 2006.

[28] A. Zisman, K. Mahbub, and G. Spanoudakis. A service discovery framework based on linear composition. *Services Computing, IEEE International Conference on*, 0:536–543, 2007.

[29] A. Zisman, G. Spanoudakis, and J. Dooley. A Framework for Dynamic Service Discovery. *2008 23rd IEEE-ACM International Conference on Automated Software Engineering*, (September):158–167, 2008.