Towards a Secure Service Coordination

Thi-Huong-Giang Vu

LSR-IMAG Laboratory, BP 72, 38402 Saint Martin d'Hères, FRANCE
Thi-Huong-Giang.Vu@imag.fr
http://www-lsr.imag.fr/Les.Personnes/Thi-Huong-Giang.Vu/

Abstract. This paper presents an approach for building secure service-based coordinated systems. Secure coordination is considered at two levels: abstraction (i.e., specification) and execution (i.e., run level). At the abstraction level, we define a general model enabling to specify coordination and its related non functional properties (such as security). The idea is to use constraints for expressing the application logic of a coordinated system and its required security strategies. Coordination activities are the key concepts used for controlling the execution of participating services. Constraints are specified as pre and post conditions of these coordination activities. At the execution level, we propose an architecture which implements strategies to verify constraints and manage the secure execution of coordination. We propose also an instantiating vade-mecum to configure execution level components according to a specific set of constraints.

1 Context and Motivations

The democratization of Internet along with recent advances in information technologies has made the global networked marketplace vision a reality. In such an environment, companies form alliances for building information systems that aggregate their respective services, and thereby enabling them to stay competitive. Effective service sharing and integration is a critical step towards developing next generation of information systems for supporting the new online economy. Given the time-to-market, rapid development and deployment requirements, information systems are made up of the services of different service providers, accessible through networks, e.g., Internet. Such information systems are called *coordinated systems*. A service performs functionalities associated with a goal desired by its provider. Services are heterogeneous, and use different data formats and transport protocols. A service provider is an autonomous organism that keeps control on the service execution with respect to some non-functional aspects such as security. A service can evolve independently of its users (applications) by both aggregating new functionalities or, conversely, removing existing ones. A service provider predefines also instructions and descriptions for using its services (e.g., where and when functionalities of these services can be accessed). Using a service implies invoking a provided method and (possibly) waiting for execution results.

Numerous systems, models and languages have been proposed for supporting service coordination, i.e., the way services invocations are orchestrated according to the application logic of a given coordinated system. Existing solutions such as workflow models [16, 7] or Petri nets [13] tackle the specification and enactment of service coordination. Using a workflow model, the execution of a coordinated system is controlled by a data flow and a control flow. The data flow specifies data exchange among participating services. The control flow describes their dependencies and it is expressed by ordering operators such as sequence, selection (OR-split, OR-joint) and synchronization (AND-split, AND-joint). Using a Petri net, the execution of a coordinated system is expressed in form of rules applied on data delivered to or consumed by participating services (i.e., places). It implies (i) rules for abstracting the structure of exchanged data (i.e., tokens) between services and (ii) rules for scheduling and firing input and output data of service execution (i.e., transitions). The execution of interaction among services has been facilitated by current technologies, such as technologies driven by the interoperation approach [5, 15] and the intercommunication approach [4].

While particular attention has been devoted to service coordination, non-functional aspects such as security have been poorly addressed by existing coordination models, languages and execution engines. It is hard to accurately specify what a coordinated system has to do under specific security requirements such as authentication, reliability, non repudiation and messages integrity. It is also often difficult to consider in advance the coordination of participating services under a large set of interactions and interdependencies among them. A loose specification of application logic can lead to a wrong order of interactions among services. We can also mistreat real situations during the coordination execution, e.g., invoked service is undesirably replaced by another. Furthermore, at execution time managing secure coordination implies:

– Authentication of the services that participate in a coordination process (i.e., identify the invoked service and the service that provides results after an invocation).
– Verifying messages integrity (i.e., those exchanged among services) in order to avoid their unauthorised alteration.
– Ensuring non repudiation of coordination: post-check the validity of coordinated system execution and prevent a participating service from denying previous actions.

The challenges are to avoid security vulnerabilities that can reach the service coordination and to provide strategies and measures for ensuring security at run-time. Moreover, the proposed strategies and measures should not contradict the facility of the coordinated system construction and the flexibility of services. It should be possible to adapt coordination and security aspects of coordinated systems on different topologies, usage scenarios, delegation requirements and security configurations. It should imply also the way to customize security levels for different types of participating services when they take part in different coordinated systems. Therefore, we aim at adding security properties service coordination.

Our approach enables secure service coordination by combining security properties of services. It also defines the general architecture of components for managing secure coordination at run-time. We assume that services and network security is ensured by the communication and the execution environments. Particularly, we suppose that there is no backdoor for accessing public instructions and descriptions exported by services; and that exchanged information confidentiality is ensured by underlying network services (e.g., by cipher mechanisms). We also assume that security properties are exported by services and that they are implemented by heterogeneous tools (e.g., different encryption algorithms).

The remainder of this paper is organized as follows. Section 2 introduces the model for secure service coordination. Section 3 presents the run-time architecture for verifying constraints and managing the secure execution of coordination. Section 4 describes the instantiating vade-mecum for programming security tools supported by the execution and communication environments. Section 5 compares our work with existing ones. Finally, section 6 concludes the paper and discusses further research directions.

2 Model for secure service coordination

We propose a model (see Fig. 1) that offers concepts to describe service coordination as coordination activities and their associated constraints.

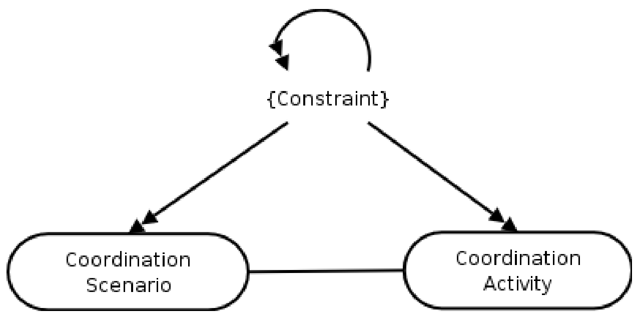

Fig. 1. Secure service coordination model

A *coordination activity* specifies an interaction between two services, where one invokes a function provided by the other and (possibly) waits for execution results.

A *coordination scenario* is the history containing information about the execution of a coordinated system. It is built by tracing the execution of coordination activities.

A *constraint* specifies the behaviour, the data, the characteristic or the interface associated to a coordination activity or to a coordination scenario. A constraint can be enabled, enforced, observed and verified.

In our model, the application logic of a service-based coordinated system is specified as a set of constraints. These constraints are added to coordination activities (in form of their preconditions, post-conditions and invariants) and refer to a given coordination scenario. In this way, an application logic can address different types of requirements imposed to the execution of coordination activities: ordering (e.g., their temporal relationships), firing (e.g., the moment in which a participating service must be invoked) and data interdependencies (e.g., input/output data relationships).

Security strategies required by a coordinated system are specified in a similar way. Constraints expressed on security properties provided by services are coupled with constraints used to control the execution of coordination activities. These constraints are also added to coordination activities and refer to a given coordination scenario. A security strategy addresses integrity, authentication, authorisation and non repudiation for coordination. The coordination can be then controlled and managed to be performed with respect to specific functional safety requirements such as in the correct time, in the correct communication cross-links, by the correct actors, etc.

Let us consider a flight booking application built out of three existing services:

– Adventurer service manages clients that are interested in booking flights.
– Payment service executes online payment transactions on given client accounts.
– Seeking service looks for available seats and performs flight pre-booking operations on a flight database.

The application logic of such a coordinated system is explained as follows. The flight booking application first interacts with Adventurer service to get information about client and her/his needs by invoking the method get_Requirements. This information is used by the method seek_Flights of the Seeking service for looking for available flights. This service returns a list of possible flights that are displayed by the method display_Results of the Adventurer service.

Constraints (ordering, firing and data interdependencies) express this application logic as pre and post conditions associated to the three following coordination activities:

– A_1: getFlightInformation(AdventurerService) where reservation requirements are retrieved.
– A_2: seekFlights(SeekingService) for looking for available flights according to the information received as input (from A_1).
– A_3: showResults(AdventurerService) for displaying the booking result.

Required security properties such as authentication and authorisation are also expressed in form of pre and post conditions of these coordination activities. Corresponding security strategies specify that these identified coordination activities are permitted for invoking their relating methods and that received results really stem from the invoked services.

For example, examine the coordination activity A_2 and its related coordination activities A_1 and A_3. The following constraints specify coordination and security aspects associated to A_2:

– Obligate(S_1 = COMMIT): once the execution status of A_1 (i.e., S_1) is succesful, the method seekFlights of the Seeking service can be invoked.
– Match(O_1): information about customer's needs (i.e., O_1) produced by the method getFlightInformation of the Adventurer service is used as input data of the method seekFlights provided by the Seeking service.
– After(getFlightInformation): the end of the execution of the method getFlightInformation must precede the beginning of the execution of the method seekFlights.
– Approved(seekFlights): the identity of the service providing the invoked method seekFlights must belong to the approved list of the coordinated system.

Similarly, the following post conditions of A_2 must hold:

– Permit(S_3 = READY): A_3 can fire the invocation to a method of the Adventurer service. This constraint plays also the role of an authorisation constraint for A_3.
– Invariant(O_2): the flight search result cannot be altered until it is redelivered to the Adventurer service.
– Received(seekFlights, invocation) $\wedge$ Sent(seekFlights, result): it ensures that the invocation and the transmission of results are done within the same execution scope. In the example, the invocation of the method seekFlights is received and its results are sent within the scope of the coordination activity A_2. This constraint is used for avoiding non-repudiation.

3 Execution manager general architecture

We propose an architecture for realising strategies to execute secure service coordination. This execution architecture provides components that can be adapted to manage security strategies to specific application requirements. It provides components for managing functional aspects and non functional aspects, in particular security. For example, at the participating services side, functional aspects are the methods they provide. In our example a security aspect of the Adventurer service is client authentication. The functional aspect of a coordinated system is its application logic and its non-functional aspects concern functional safety.

We define execution managers, which consist of control components that are associated to services and coordination activities. It is extended with other trusted third-party components. For a given coordination activity, each participating service is associated to a connector and a partner controller. A coordination activity is executed by a builder and it has an associated security controller that supervises its execution and the execution of the related participating services. There are three types of security controllers: constraint builders, strategy controllers and aspect controllers. Security properties of services are homogenised by wrappers which are also responsible of managing secure interaction between services.

3.1 Coordination activity builder

Fig. 2 shows the components that execute coordination activities.

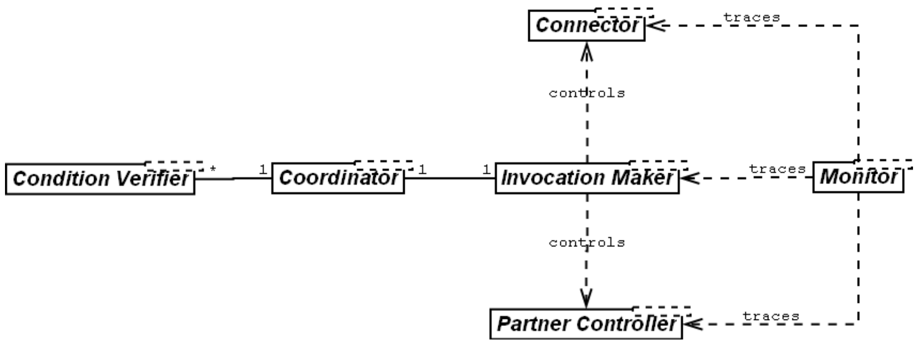

Fig. 2. Components for implementing coordination activities

A *coordinator* controls the execution of a coordination activity with the support of an *invocation maker* and *condition verifier*s. It builds a schedule that specifies the moment in which one or several constraints must be verified with respect to the invocation of a method.

An *invocation maker* performs the invocation of a method specified within a coordination activity.

A *partner controller* manages the interaction with the service related to a specific invocation.

A *connector* is used as a specific communication channel for the interaction between the partners of an invocation.

A *monitor* traces the execution of these components and notifies the execution status to build the coordination scenario.

3.2 Constraint builder

Fig. 3 shows the components used for managing constraints: *constraint solver*s, *exception handler*s and *condition verifier*s.

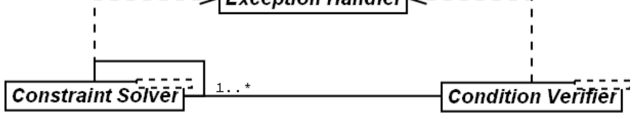

Fig. 3. Constraint builder components

A *constraint solver* verifies a set of constraints. It returns a Boolean result. The *false* value is considered as an exception and it is managed by another component.

An *exception handler* manages exceptions raised within the execution of a coordinated application. It collects the execution results of other components an generates information associated to the coordination scenario.

A *condition verifier* combines the results of constraint solvers to evaluate pre and post conditions of a specific coordination activity. It translates invariants to equivalent pre and post conditions. All preconditions are checked before launching an invocation. All post conditions are checked after receiving the result from the corresponding invocation.

3.3 Strategy controller

Fig. 4 shows the components that implement strategies associated to a coordination.

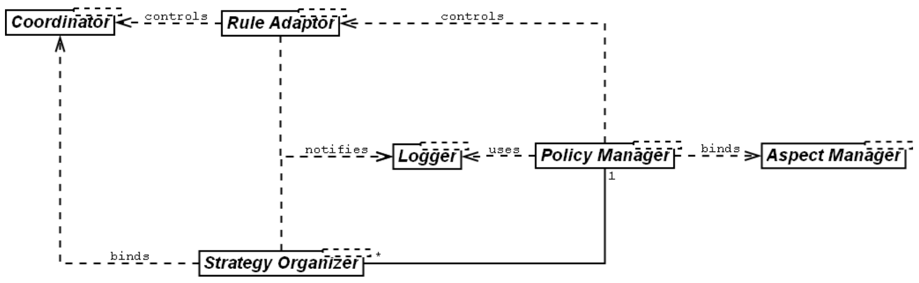

Fig. 4. Strategy controller components

A *rule adaptor* schedules coordination activities according to pre-defined strategies. It controls the execution of a set of coordination activities.

A *strategy organizer* implements a strategy specified by constraints using properties of the participating services.

A *policy controller* controls and manages the execution of the strategies for a given coordinated application.

A *logger* collects information about the execution state of strategies and the notifications from monitors, exception handlers and stores them in a log.

3.4 Aspect controller

Fig. 5 shows the components for managing security policies of participating services.

Aspect form generator describes security properties of a participating service and exports it.

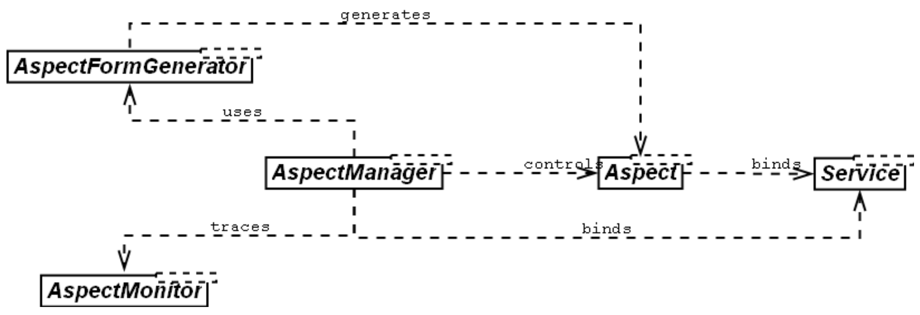

Fig. 5. Aspect controllers - wrappers for participating services

Aspect manager manages security properties of a service.

Monitor traces modifications on security properties and matches properties with application requirements.

4 Instantiating vade-mecum

We describe an instantiating vade-mecum for programming security tools supported by the execution and communication environments (e.g., secure data exchange among services). Our instantiating vade-mecum assists programmers to configure the components of a service-based coordinated system according to a specific set of constraints. The vade-mecum helps to avoid conflicts and redundancy of coordination and security strategies supported by the components of the proposed architecture and those supported by real environment. The vade-mecum provides a technical catalogue of security strategies that can be associated to given coordination contexts, and a guide for combining security strategies with given coordination constructors.

Reinforce coordination rule. Security constraints associated to a coordination activity are explicitly specified in our model. Those associated to participating services are implicit to the description of such services. For giving supplementary effects of protection at run-time (e.g., for supervising exchanged information and for notifying the potentially dangerous scripts), security strategies which are not (or implicitly) specified by constraints can be implemented and instantiated as an instance of a *Strategy Organizer*.

Establish privileges for actors related to a coordination activity. Authorisation constraints associated to a coordination activity specify in which conditions such an activity can be executed. The role of an actor plays and his/her associated privileges are important elements for verifying constraints. We describe how to grant and manage invoking privileges, and how to associate them to the corresponding connectors.

Supervise coordination activity orchestration. We describe how to construct an activity state-transition automaton for pre-checking the causality of coordination activities for a given set of participating services.

5 Related Works

Existing works can be classified into two categories according to their service interaction mechanisms. In the first category, services interact through a shared space [6, 8, 1]. Security policies are associated to the shared space: access control (authorisation, control privilege, etc.) and services authentication. Target coordinated system configuration is specified by suitable coordination languages, e.g., the Linda family [11, 12, 14].

In the second category, coordination is based on data exchange among participating services. Services are considered black box processes that produce and consume data via well defined interfaces. Services communicate directly for establishing connections, exchanging data, diffusing control events among processes [10, 8, 2].

In [1] security strategies are applied to tools and the environments that support interconnection and communication among participating services (i.e., only at coordination execution level). WS-Policy and WS-Secure-Conversation combined with WS-Security and WS-Trust are going in this direction. [6] presents an approach for building a secure mobile agent environment. [3] specifies secure exchanged messages among Web services based on SOAP protocols. [9] proposes a formal security model to identify and quantify security properties of component functionalities to protect user data by evaluating and certifying the components and their composition.

6 Conclusion and Future Work

This paper presented our approach towards secure service coordination. We described our coordination model and an associated architecture for addressing services authentication.

In conclusion, the main contribution of our work is to provide secure service coordination by specifying security strategies and an associated architecture for executing them. Security properties provided by services are homogenized under a pivot view that can be used for specifying well suited security strategies according to specific requirements.

We are currently specifying and implementing a secure coordination framework called MEOBI. Future work includes evaluating MEOBI for component-based and Web services based systems. Further research focuses on the extension of secure coordination strategies by including performance requirements.

Acknowledgment

This research is supported by the French National Research Council (CNRS). I would like to thank Prof. Christine Collet and Dr. Genoveva Vargas-Solar for their support, guidance and attentive comments, Dr. Christophe Bobineau for his constructive discussions during the preparation of this paper. I would like to extend my appreciation to the reviewers for their feedbacks on the previous draft of this paper.

References

1. Alvarez, P., Banares, J.A., Muro-Medrano, P.R.,Nogueras, J.,Zarazaga, F.J.: A Java Coordination Tool for Web-service Architectures: The Location-Based Service Context. In *FIDJI'01: Revised Papers from the International Workshop on Scientific Engineering for Distributed Java Applications*, Springer-Verlag (2003) 1–14
2. BEA Systems, IBM Corporation, Microsoft Corporation: Web Services Coordination (2003)
3. Belhajjame, K., Vargas-Solar, G., Collet, C.: Defining and coordinating open-services using Workflow. In: *Proceedings of the Eleventh International Conference on Cooperative Information Systems, Lecture Notes in Computer Science* (2003)
4. IBM Corporation: http://www-306.ibm.com/software/htp/cics/ (1999)
5. Object Management Group: http://www.corba.org/ (2002)
6. Cremonini, M., Omicini, A.,Zambonelli, F.: Coordination in Context: Authentication, Authorisation and Topology in Mobile Agent Applications. In: *3rd International Conference on Coordination Languages and Models*, Springer-Verlag (1999)
7. Georgakopoulos, D., Hornick, M.F., Sheth, A.P.: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases* (1995) 119-153
8. Issarny, V., Bidan, C., Saridakis, T.: Characterizing Coordination Architectures According to Their Non-Functional Execution Properties. In: *31st IEEE International Conference on System Science* (1998)
9. Khan, K.M., Han, J.: A Security Characterisation Framework for Trustworthy Component Based Software Systems. In: *IEEE International Computer Software and Applications Conference* (2003)
10. Klint, P., Olivier, P.: The TOOLBUS Coordination Architecture: A Demonstration. In: *5th International Conference on Algebraic Methodology and Software Technology*, Springer Verlag (1996) 575–578
11. Malone, T.W.: What is Coordination Theory and How Can it Help Design Cooperative Work Systems?. In *CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work* (1990) 357–370
12. Papadopoulos, G.A., Arbab, F.: Coordination models and languages. *Advances in Computers* (1998)
13. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall PTR (1981)
14. Tolksdorf, R.: Coordination Technology for Workflows on the Web: Workspaces. In *COORDINATION '00: Proceedings of the 4th International Conference on Coordination Languages and Models*, Springer-Verlag (2000) 36–50
15. Microsoft Corporation: http://msdn.microsoft.com/webservices/building/interop/ (2003)
16. Workflow Management Coalition: Terminology and Glossary (1996)