From Goals to Reliable Service Compositions*

Liliana Pasquale
Politecnico di Milano - Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci, 32 - 20133 Milano, Italy
pasquale@elet.polimi.it

Abstract

A key feature of modern enterprises lies in the availability of software systems able to adapt themselves to the frequent changes in the business processes. Services have already proven their ability to provide flexible solutions, but so far the focus has been mainly on the technological infrastructure. Oftentimes user requirements have been neglected and also the reliability of proposed solutions has been traded for flexibility and dynamism. These are the motivations for the research proposal presented in this paper. Starting from the actual user requirements, we aim to provide a complete solution to (semi) automatically derive coherent, complete, and reliable service compositions. The proposal uses a goal model to represent the business goals and supervision directives to oversee and enforce the reliability of obtained compositions. Execution and supervision are supported through a flexible run-time infrastructure.

1 Research Problem

It is already proven that service-based IT solutions provide different advantages [2]: quick reaction and adaptation to new needs, easy integration of heterogenous components, and reduced development costs. So far, these solutions have been mainly addressed from a technological perspective, but, in contrast, they should come from the actual requirements of the different stakeholders. Besides the technological achievements, we need methods and tools to precisely relate requirements to services and service compositions (also known as processes).

The business dimension imposes frequently changing requirements, while the distributed nature makes them intrinsically unreliable: network failures may happen, partner services can be down or unavailable, or they can change independently of the applications that use them. This means that requirements must state the functionality of the system-to-be, its qualities of service, by means of proper KPIs[1] (Key Performance Indicators), and also how to cope with changes in both the requirements and the execution environment (self-adaptation capabilities). Functional requirements mainly dictate the service composition, while non-functional ones help define the KPIs of interest and how to keep the system on track in case of anomalies.

The general goal of releasing reliable service compositions, which meet user requirements and remain aligned with them, poses challenges at different levels of abstraction. At requirements level, we must provide suitable models to represent requirements, along with their potential changes. These requirements must also be related to the corresponding implementations to assess their satisfaction and be able to propagate their changes onto the system. At application level, KPIs must be constantly measured and self-adaptation capabilities be activated when needed to keep the execution on track and give the user the perception of a reliable and trustable solution.

Available models and technologies are not able to completely address the issues discussed above. Goal models [8, 21] and scenario-based solutions do not provide explicit support to uncertainty or adaptation. Moreover, common business process notations, like BPMN (Business Process Modeling Notation [13]), are not expressive enough to embed requirements into the business process, trace them, and account for user-oriented reliability.

The proposal presented in this paper starts from powerful models to enforce requirements in service compositions from application design down to execution. It assumes the adoption of a *live* goal model able to change at runtime and track the satisfaction level of its requirements. Requirements traceability is guaranteed by linking the goal model to both a functional and a supervision model of the service composition. The former represents the set of compositions able to satisfy stated requirements. The latter defines how to

*This research has been partially funded by the European Commission, Programmes: IDEAS-ERC, Project 227977 SMScom, and FP7/2007-2013, Projects 215483 S-Cube (Network of Excellence) and 216556 SLA@SOI.

[1] KPIs provide business-oriented indicators of the actual performance of the system.

assess the KPIs of interest and keep the application on track (i.e., how to add reliability to service compositions). A flexible runtime infrastructure supports the controlled execution of service compositions through the adoption of suitable engines, along with data collectors, monitors and adaptors to enact the directives defined by the supervision model.

2. Background and Related Work

The background of our proposal comprises solutions for modeling the requirements of service-based, and adaptive, systems, tracing them, and overseeing the reliability of compositions.

As for the first aspect, some preliminary works [5] have already used goal models for specifying the requirements of adaptive systems. Their adaptation strategies are expressed as enumerations of predefined alternative tasks. Our solution, in contrast, aims to provide means to specify adaptation solutions that can evolve with respect to the actual execution of the system and its past history.

The bridge between requirements and service compositions has already been studied by Lo and Yu [9], who relate business patterns to "recurring" service-oriented solutions, and by Pistore et al. [7]. They are interested in the off-line formal verification of goal satisfaction and relate process activities to their corresponding goals through annotations. These are interesting works, but we would like to obtain more from requirements. We would like to be able to infer the structure of the service composition and, when possible, derive the process activities and their execution order directly from goal operationalization. We also want to use goals to infer a coherent supervision model that oversees the execution of the service composition and helps keep it on track.

Also the problem of requirements monitoring has already been addressed. Mylopolus et al. [19] use the generation of log data to infer the denial of requirements and detect problematic components. Robinson [16] distinguishes between the design time model, where business goals and their possible obstacles are defined, and the runtime model, where logical monitors are derived from the obstacles and are applied onto the running system. Obstacle analysis helps detect possible problems. Other monitoring approaches do not start from requirements and cover different functional aspects and qualities of services. For example, Ludwig et al. [10] monitor the run-time state of the agreements established between a service provider and its customers, while Erradi et al. [4] propose a middleware to support the fault-tolerant execution of services based on user-specified policies.

These proposals, and others [17, 15], only concentrate on overseeing the different aspects related to execution. As for adaptation, Dynamo [6] provides atomic recovery actions, which can be combined to create user-defined strategies, to adapt the behavior of the different process instances, while VieDAME [11] and SCENE [3] use dynamic binding techniques to change bound services at runtime.

Our proposal builds on these solutions and emphasizes the separation among probing, analysis, and reaction to provide a single homogeneous framework in which the different approaches can be seamlessly integrated. Different solutions cover different aspects; their integration leads to a complete, but customizable, solution without reimplementing everything in a single assembly. This way, monitoring and adaptation capabilities are not hard-coded in the infrastructure, but can be selected and customized according to the actual needs.

3. Proposed Approach

Figure 1 sketches our approach. It adopts a goal model to represent the requirements (WHY). From the goals we derive a functional and a supervision model of the service composition (WHAT). Finally we support its reliable execution through a suitable run-time infrastructure (HOW).

The proposal exploits the KAOS goal model [18] as starting point to represent requirements[2]. This choice lays in the ability of goals to show the alignment of the system-to-be with the organization's objectives through refinement relationship among goals [18]. We also plan to combine KAOS with goal modeling approaches for self-adaptive systems, like RELAX [20], to express uncertainty. RELAX distinguishes between critical and non critical goals. The latter may be relaxed with an alternative definition that must hold in critical situations to ensure the former.

The capability of automatically deriving goals' operations, as explained by Letier [8], eases the derivation of the functional model and permits us to automatically derive the supervision directives that must be applied at runtime. We also argue that the relaxed definition of non-critical goals can guide the generation of recovery actions that modify the flow of process activities when critical situations are encountered.

The functional model adopted to specify the behavior of service compositions is based on an abstract version of BPEL (Business Process Execution Language [12]). It allows us to embed in a single specification the different process implementations that satisfy defined requirements. The method applied to infer the functional model is based on the following macro steps:

- We associate each operation with an abstract service composition fragment and the set of its partner services. Process activities are derived by inspecting the

[2]Lack of space does not allow us to provide details about KAOS. We assume that readers are familiar with its main concepts.

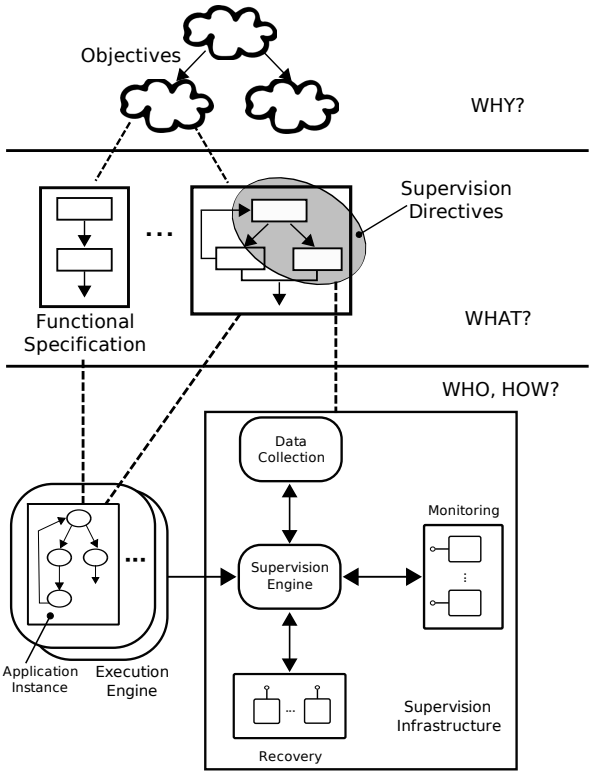

Figure 1. Overall approach.

definition of the corresponding operations. The events adopted in the definition of an operation are transformed into activities that imply the interaction of the process with a partner service. For example, events that appear in preconditions are associated with BPEL *receive* or *pick* activities, while events involved in post-conditions generate *invoke* or *reply* activities. We plan to develop general strategies to derive the methods partner services are supposed to provide and group them into suitable interfaces.

- We compose process fragments according to the criteria imposed by the pre- and post-conditions of the operations process fragments are associated with. Since the generation of functional specifications automatically is not always possible, human intervention may be needed.

Along with the functional specification, we provide a supervision model in the form of data collection, monitoring and recovery directives. Data collection directives define the data on which monitoring constraints are verified or information exploited by recovery strategies. They have a direct mapping onto the data that can be collected at runtime: process internal variables, exchanged messages, data obtained by invoking external probes, and data coming from previous process executions. Monitoring directives specify the behaviour and KPIs the process must provide. They can be represented in terms of FOL (First Order Logic) or through LTL (Linear Temporal Logic) assertions over a process scope. Recovery directives are expressed in a way similar to [6]. They are complex strategies composed of atomic actions: for example, retry (retries to invoke a partner service), notify (informs a user about a problem), substitute (changes a faulty partner service with another one). Some of them can only be applied synchronously while others are asynchronous. Recovery actions must indicate their scope: the process instance that is being monitored, a subset of the running process instances, or the actual process definition and thus all its future instances. In this last case, we also need to manage, and make coexist, the different versions of the process definition.

Supervision directives can be defined manually. In this case, we should provide an intuitive language to ease the definition of the reliability directives and hide their complexity. This activity may be risky, since recovery defined by users may conflict with set goals or with other recovery directives already defined. Our proposal is to investigate patterns to monitor basic QoS properties, compose KPIs, and manage recurring failures. For example, if the monitoring assertion is about response time, we can exploit a recovery directive that invokes an alternative service if the first one is not available. We would like to derive supervision directives (semi) automatically from goals. The formalization of goals allows us to infer the corresponding monitoring directives, expressed either through pre- and post-conditions or through temporal constraints. The scope of these constraints is derived from the process activities the goal relies on. Data collection directives are retrieved by looking at the elements (process variables, messages, etc.) that correspond to the events and entities adopted in the goal's specification.

Finally we will provide a flexible runtime infrastructure able to support the reliable execution of processes and enforce supervision directives. Our preliminary ideas are described in [1]: the execution infrastructure controls different BPEL engines integrated over the same set of supervision components. The architecture is centered around plug-ins to easily add and remove different data collector, monitoring, and recovery components. Each *BPEL Engine* is an instance of ActiveBPEL Community Edition Engine augmented with probes to collect process data. The *Data Collector* acts as façade for collecting external data and for retrieving and storing historical data from/in the repository. The *Monitoring* and *Recovery* components hold the monitoring and recovery plug-ins, respectively, we want to use.

The main element of the architecture is the *Supervision Engine*, based on rule engine technology [14]. It is in charge of activating external and historical data collection, as well as any monitoring and recovery activity. This is achieved by

defining rules on the data contained in the working memory: process state data coming from probes, external and historical variables collected through the *Data Manager*, and analysis results. Our framework also provides a configuration tool called *Supervision Manager*, in charge of configuring the various components of the framework according to the reliability model of interest. In particular, it configures the probes within the processes, to collect internal data, the *Supervision Engine*, to retrieve external and historical data, and defines how monitoring and recovery are activated. It also provides the *Monitoring* component with the constraints each plug-in is supposed to check and the *Recovery* component with the recovery strategies.

References

[1] L. Baresi, S. Guinea, and L. Pasquale. Integrated and Composable Supervision of BPEL Processes. In *Proc. of the 6th Int. Conf. of Service Oriented Computing*, pages 614–619, 2008.

[2] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, and R. Shah. *Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap*. Prentice Hall PTR, 2005.

[3] M. Colombo, E. D. Nitto, and M. Mauri. SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In *Proc. of the 4th Int. Conf. on Service Oriented Computing*, pages 191–202, 2006.

[4] A. Erradi, P. Maheshwari, and V. Tosic. Policy-Driven Middleware for Self-adaptation of Web Services Compositions. In *Proc. of the 7th Int. Middleware Conf.*, pages 62–80, 2006.

[5] H. J. Goldsby, P. Sawyer, N. Bencomo, B. H. C. Cheng, and D. Hughes. Goal-Based Modeling of Dynamically Adaptive System Requirements. In *Proc. of the 15th Int. Conf. on Engineering of Computer-Based Systems*, pages 36–45, 2008.

[6] S. Guinea. *Dynamo: a Framework for the Supervision of Web Service Compositions*. PhD thesis, Politecnico di Milano, 2007.

[7] R. Kazhamiakin, M. Pistore, and M. Roveri. A framework for integrating business processes and business requirements. In *Proc. of the 8th Int. Enterprise Distributed Object Computing Conf.*, pages 9–20, 2004.

[8] E. Letier. *Reasoning about Agents in Goal-Oriented Requirements Engineering*. PhD thesis, University of Louvain, Belgium, 2001.

[9] A. Lo and E. Yu. From business models to service-oriented design: A reference catalog approach. In *Proc. of the 26th Int. Conf. on Conceptual Modeling*, pages 87–101, 2007.

[10] H. Ludwig, A. Dan, and R. Kearney. Cremona: an architecture and library for creation and monitoring of WS-agreents. In *Proc. of the 2nd Int. Conf. on Service Oriented Computing*, pages 65–74, 2004.

[11] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive Monitoring and Service Adaptation for WS-BPEL. In *Proc. of 17th Int. World Wide Web Conf.*, pages 815–824, 2008.

[12] OASIS. Business Process Execution Language for Web Services, Version 1.1. BPEL4WS specification, 2003.

[13] OMG - Object Management Group. BPMN. `http://www.bpmn.org/`.

[14] M. Proctor and et al. Drools. `http://www.jboss.org/drools/`.

[15] F. Raimondi, J. Skene, and W. Emmerich. Efficient online monitoring of web-service slas. In *Proc. of the 16th ACM SIGSOFT Int. Symposium on Foundations of software engineering*, pages 170–180, 2008.

[16] W. N. Robinson. Monitoring Web Service Requirements. In *Proc. of the 11th Int. Requirements Engineering Conf.*, pages 65–74, 2003.

[17] SeCSE Integrated Project. A4.D8 Policy Specification and Integration with Existing Standards. Technical report, 2006.

[18] A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[19] Y. Wang, S. A. Mcilraith, Y. Yu, and J. Mylopoulos. Monitoring and Diagnosing Software Requirements. *Automated Software Engg.*, 16(1):3–35, 2009.

[20] J. Whittle, P. Sawyer, N. Bencomo, and B. H. C. Cheng. Relax: Incorporating uncertainty into the specification of self-adaptive systems. Technical report, 2009.

[21] E. S.-K. Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, Toronto, Ont., Canada, 1996.