Related Work and Summary

Related work concerning the enactment, run-time verification, monitoring and mining of interaction models is presented. Then, the major contributions of this part of the book are briefly summarized.

16.1 Related Work

We separate the discussion of related work into three areas:

- run-time verification and monitoring;
- enactment;
- process mining.

16.1.1 Run-Time Verification and Monitoring

Run-time verification and monitoring are open and challenging research topics, especially in the Service Oriented Computing setting. In [36], the authors present a broad survey of principles, techniques, and methodologies proposed in the literature for the monitoring and adaptation of (web) services. In particular, they classify the state of the art approaches along two different dimensions: purpose (why) and target (what) of the monitoring task.

Concerning the purpose of monitoring, they identify six areas:

- fault monitoring, checking whether the system conforms to some specification, and identifying correct and noncompliant situations;
- monitoring as a first step towards diagnosis and recovery of faulty services;
- monitoring as part of a framework addressing the optimization of resource allocation;
- monitoring for triggering and driving the dynamic adaptation of services;
- Business Activity Monitoring, constantly measuring and evaluating key performance indicators and business rules;
monitoring as part of a framework dealing with the long-term adaptation of services, taking into account both present and historical information.

For what concerns the targeted information, monitoring approaches can be roughly classified into two groups: monitors dealing with functional and behavioral properties (such as pre and post-conditions in service calls, constraints over the messages exchange, and data-related properties), and monitors targeting non-functional properties (such as performance, reliability, and response time). Finally, monitoring applications distinguish themselves by their abstraction degree: some approaches are implementation-dependent, and focus on specific architectures and languages, whereas other approaches are abstract, and rely on business-level concepts abstracting away from implementation and technological details.

In [27], Baresi and Guinea present an approach towards the dynamic monitoring of BPEL-based services. In particular, the BPEL specification of interest is annotated with WS-Col (Web Service Constraint Language) assertions, mainly focusing on QoS aspects. A code weaving preprocessing procedure is then used before the BPEL process is deployed. The preprocessing phase parses the BPEL specification, analyzes the monitoring assertions and introduces further activities dedicated to send the relevant informations to a monitoring manager, where needed. WS-Col properties are verified against each instance of the BPEL process, and can contain expressions referring to the actual values of data. During the execution, the monitoring manager dynamically decides whether some rule must be currently checked. If so, it acts as an intercepter: when a pre-condition rule must be checked, it processes the incoming message before the target BPEL process, whereas when a post-condition rule must be verified, it intercepts the message sent by the BPEL process; after having verified the property, it takes care of forwarding the message to the right recipient.

The approach has been further elaborated in [28], where a complete architecture based on Aspect Oriented Programming is presented. The resulting framework, called Dynamo, does not only provide monitoring facilities, but supports also recovery strategies.

In [26], Barbon et al. propose a JAVA-based framework for monitoring service orchestrations implemented in BPEL. A monitor intercepts all the incoming and outgoing messages exchanged by the controlled BPEL orchestration, without needing to change its implementation. Thus, differently from [27], the monitoring facility and the process are kept completely separated, and their execution engines run in parallel on the same application server. Two kinds of monitors are supported:

- instance-level monitor, targeting a single instance of the BPEL process;
- class-level monitor, aggregating information from the running process instances.

Finally, the authors introduce the Run-Time Monitor specification Language (RTML), a past-tense LTL language which accommodates data, time