Chapter 1

QUALITY ANALYSIS OF DEPENDABLE INFORMATION SYSTEMS

Apostolos Zarras and Valerie Issarny
INRIA UR Rocquencourt
Domaine de Voluceau
78153 Le Chesnay
France
{Apostolos.Zarras, Valerie.Issarny}@inria.fr

Abstract Large industrial organizations strongly depend on the use of enterprise information systems for the application of their complex business processes. Typically, an enterprise information system (EIS) consists of a set of autonomous distributed components providing basic services. Business processes can be realized as workflows consisting of: (1) tasks combining basic services provided by EIS components and (2) synchronization dependencies among tasks. EIS users have ever-increasing non-functional requirements (e.g. performance, reliability, availability, etc.) on the quality of those systems. To satisfy those requirements, EIS engineers must perform quality analysis and evaluation, which involves analytically solving, or simulating quality models of the system (e.g. Markov chains, Queuing-nets, Petri-nets etc).

Good quality models are hard to build and require lots of experience and effort, which are not always available. A possible solution to the previous issue is to build automated procedures for quality model generation. Such procedures shall encapsulate previous existing knowledge on quality modeling and their use shall decrease the cost of developing quality models. In this paper, we concentrate on the performance and reliability of EISs and we investigate the automated generation of quality models from EIS architectural descriptions comprising additional information related to the aspects that affect the quality of the EIS.

Keywords: Performance, Quality, Reliability, Software Architecture, Workflow.

P. Ezhilchelvan and A. Romanovsky (eds.), Concurrency in Dependable Computing.
© 2002 Kluwer Academic Publishers. Printed in the Netherlands

1. Introduction

Today's industrial organizations use large scale enterprise information systems for performing and managing their complex business processes. An enterprise information system (EIS) typically is built of numerous, disparate, autonomous subsystems, named EIS components hereafter. Business processes can then be realized as workflows. A workflow consists of: (1) tasks combining basic services provided by EIS components and (2) synchronization dependencies among tasks [11, 7, 10]. The business processes that need to be supported by an EIS serve as the primary functional requirements for developing and maintaining the EIS. However, nowadays, non-functional requirements on the quality of the EIS (e.g. performance, reliability, availability) are also of significant importance. EIS architects, designers and developers are supposed to design, implement and maintain the EIS while taking into account the user's non-functional requirements. Consequently, quality analysis is required during the life-cycle of the EIS.

The analysis of certain quality attributes (e.g. performance, reliability, availability) is not a new challenge since a variety of techniques have been proposed and used for several years [4, 5]. Those techniques are supported by an underlying modeling formalism, which allows to specify structural and behavioral aspects of the inspected system that affect the system's quality. Well known examples of such formalisms are block diagrams, graphs, Markov chains, Petri-nets, Queuing-nets, logics, etc. The resulting models are then analytically solved, or simulated. Based on the above, the challenge nowadays becomes to make existing techniques more tractable to the end users. The main problem today is that building good quality models, which when solved or simulated, give accurate predictions on the quality of the system, requires lots of experience and effort. EIS architects, designers and developers use architecture description languages (ADLs) and object oriented notations (e.g. OMT, UML) to design the EIS architecture. It is a common case that they are not keen on building quality models using Markov chains, Petri-nets, Queuing-nets etc.

Hence, the ideal approach would be to provide the EIS architects, designers and developers with an environment, which enables the specification of EIS architectures and further provides adequate tool support for the automated generation of models suitable for the quality analysis of the system. In this paper, we investigate this issue. More specifically, we focus on the automated performance and reliability analysis of EIS and our main objectives are summarized in the following two points:

- The provision of support for modeling at the architectural level, aspects that affect the performance and reliability of EIS.
- The design and realization of automated procedures for generating traditional performance and reliability models starting from EIS architectural descriptions. The key to achieve this point is to formally specify the mapping between EIS architectural models and traditional models for performance and reliability analysis.

The remainder of this paper is structured as follows. Section 2 presents previous work related to the quality analysis of systems and identifies problems that are addressed by the approach proposed in this paper. Section 3 provides the definition of a base architectural style for specifying EIS architectures. Sections 4 and 5 detail the automated procedures for the generation of traditional performance and reliability models. Finally, section 6 concludes this paper with a summary of our contribution.

2. Background and Related Work

Pioneer work on modeling and analyzing the quality of software systems at the architectural level includes attribute-based architectural styles proposed in [3]. In general, an architectural style includes the specification of types of basic architectural elements (e.g. pipe and filter) that can be used for specifying a software architecture. Moreover, an architectural style includes the specification of constraints on using those basic architectural elements and patterns describing the data and control interaction among them. An attribute-based architectural style (ABAS) is an architectural style, which additionally provides modeling support for the analysis of a particular quality attribute (e.g. performance, reliability, availability). More specifically, an ABAS provides support for specifying:

- *Quality attribute measures* characterizing the quality attribute (e.g. the probability that the system correctly provides a service for a given duration, mean response time).
- *Quality attribute stimuli*, i.e., events affecting the quality attribute of the system (e.g. failures, service requests).
- *Quality attribute parameters*, i.e., architectural properties affecting quality attribute of the system (e.g. faults, redundancy, thread policy).
- *Quality attribute models*, i.e., traditional models that formally relate the above elements (e.g. a Markov model that predicts reliability based on the failure rates and the redundancy used, a Queuing

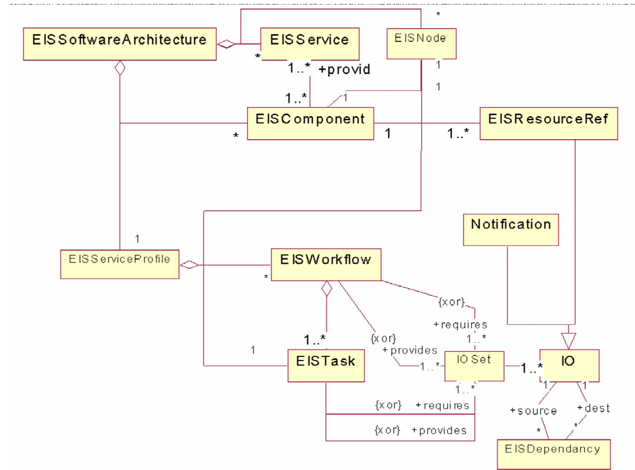
network that enables predicting the system's response time given the rate of service requests and based on the performance parameters).

In [2] the authors propose an architecture tradeoff analysis method (ATAM) where the use of an ABAS is coupled with the specification of a set of scenarios, which roughly constitutes the specification of a service profile. ATAM has been tested for analyzing qualities like performance, availability, modifiability, and real-time. In all those cases, quality attribute models (e.g. Markov models, queuing networks etc.) are manually built given the specification of a set of scenarios and the ABAS-based architectural description. However, in [2], the authors recognize the complexity of the aforementioned task. Moreover, it is our opinion that the need to manually produce quality attribute models significantly decreases the benefits of using a disciplined method such as ATAM for analyzing the quality of software systems. ATAM is a promising approach for doing things right. Nowadays, however, there is a constant additional requirement for doing things fast and easy. Asking EIS engineers to build performance and reliability models from scratch is certainly a drawback towards achieving this objective. To deal with this drawback, this paper proposes automating the generation of quality attribute models from architectural descriptions. To accomplish this goal, there is a need for specifying the mapping between architectural descriptions and quality attribute models. Hence, we need more formal definitions of ABAS. Indeed, it is not feasible to generate traditional quality attribute models starting from scenarios described in natural language and architectural descriptions within which the relationships among basic architectural elements and quality attribute measures, parameters and stimuli are not precisely defined.

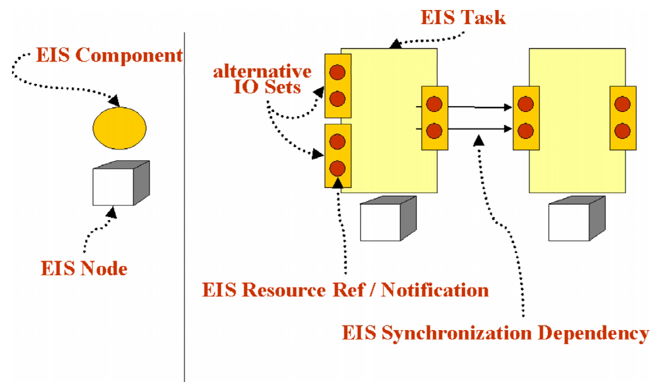
Based on the previous remarks, in the following section we present the definition of a base architectural style for the specification of EIS architectures. Then, in sections 4 and 5 we detail the mapping between EIS architectural models and performance and reliability quality attribute models.

3. A Base EIS Architectural Style

Figure 1(a) gives the UML definitions of the meta-elements used for the specification of EIS architectural models. The semantics and the use of those elements are further discussed in the remainder of this section. Moreover, Figure 1(b) gives the graphical representations of the EIS meta-elements.



(a) UML definitions of meta-elements for the specification of EIS architectures.



(b) Graphical representations of the EIS meta-elements.

Figure 1. The structure and representation of an EIS architectural model.

3.1. Basic EIS architectural elements

An EIS software architecture comprises the specification of a non empty set of EIS components and the specification of a non empty set of EIS services. An EIS component provides one, or more of the EIS services, and an EIS service is provided by at least one EIS component. Every EIS component is associated with an EIS node, representing the execution platform on top of which it is deployed. Technically, EIS components and services are specified textually using Darwin-like notations. Darwin is among the first and most popular ADLs (for more information see [8]).

An EIS architectural description further includes the specification of a service profile. A service profile is a non-empty set of EIS workflows describing how the EIS is used. An EIS workflow is a model that specifies the coordination of a set of EIS tasks. The workflow model we use is inspired by the one proposed in [11], which has recently become an OMG standard [10]. Tasks combine basic EIS services provided by components. More specifically, a task requires using a set of alternative input-sets. An input-set consists of inputs, which may be either references to EIS components, or notifications from other tasks. While executing, a task uses one of the alternative input-sets to produce an output-set (i.e. a set of references to components, or notifications to other tasks). By definition, a set of alternative output-sets may be produced by the task. Task coordination is specified in terms of synchronization dependencies among inputs and outputs. A task may be compound representing a workflow model. More specifically, a compound task consists of sub-tasks and synchronization dependencies between sub-tasks. The input and output sets of a compound task are mapped on input and output sets of the sub-tasks.

Technically, tasks and synchronization dependencies among them are specified textually using the language detailed in [11]. Moreover, textual specification of tasks describe the way different alternative input sets are used to produce the corresponding alternative output sets.

To facilitate the specification and quality analysis of EIS architectures, we developed a prototype tool whose use is demonstrated in the following subsection. The tool allows both the graphical and textual specification of EIS architectures. Already existing parsers for the Darwin and the workflow specification languages are then used for verifying the correctness of those specifications.

3.2. Example

The use of the tool for the specification and quality analysis of EISs has been tested with a real world case study, part of which we use here as an example. The goal of case study is the quality analysis of an EIS used for managing the Bull SA organization. The basic EIS architecture consists of a variety of autonomous and disparate components including:

- The `Log` component, which provides access to log files produced by a firewall system used by Bull.

- The `Billing` component, which provides billing services for Bull employees and customers.

- The `Department` component, which provides access to the personal records of Bull employees and customers.

The EIS service profile includes, among others, a workflow which combines services provided by the aforementioned EIS components into a complex billing service. The workflow consists of the following tasks:

- The `Bill` task, which uses services of the `Log` and the `Department` server to produce per-customer bills.
- The `Payment` task, which takes as input a bill produced by the `Bill` task and a reference to the `Department` server, and checks whether the bill is accepted, or not.
- The `Transfer` task, which is activated for all accepted bills and eventually uses the `Billing` server to transfer money from the account of the customer to a bank.
- The `Claim` task, which is activated for all rejected bills and uses the `Billing` server to cancel them.

Figure 2, gives a snapshot of the tool we developed showing the specification of the complex billing service workflow.

4. Automated Performance Analysis

The basic performance measures used to characterize the execution of EIS workflows and tasks are given in Table .1. Moreover, the basic stimuli that causes changes on the values of those measures is the initiation of workflows. Hence, an EIS workflows is further associated with an attribute whose value gives the statistical pattern by which the workflow is initiated. Finally, EIS components are characterized by their thread and scheduling policies, their capacity and the work demands needed for providing the associated EIS services. In the remainder, we present how EIS architectural descriptions including the specification of the previous properties can be mapped to traditional performance models.

4.1. Mapping EIS models on traditional performance models

For EIS performance analysis, we use a tool-set, called QNAP2 [1], providing a variety of both analytic and simulation techniques. QNAP2 accepts as input a queuing network model of the system that is to be analyzed.

The general structure of a queuing network model is given in Figure 3. A queuing network model consists of a set of stations providing services

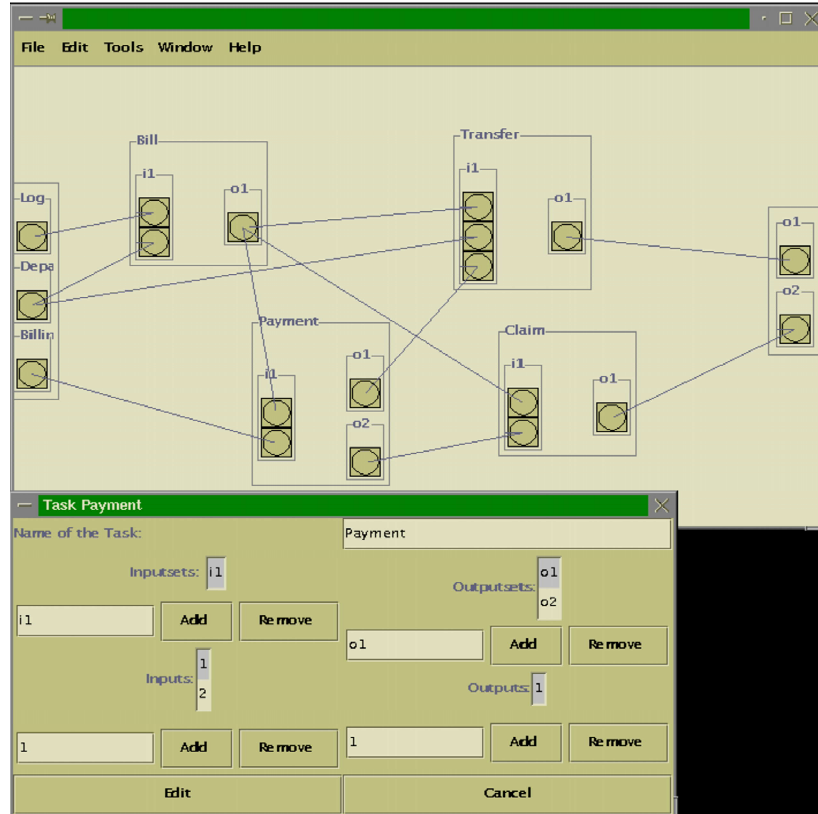


Figure 2. The specification of the complex billing service.

requested by customers. A service is associated with a set of transition rules describing what happens to a customer after the customer is served. A station is further associated with queues that store requesting customers. In a queuing network, we may have special stations, called source stations, whose purpose is to create new customers. Those stations are characterized by a statistical pattern according to which they generate customers.

Given an EIS architectural description the steps for mapping it to the corresponding queuing network are the following. First, a set of stations is generated, corresponding to EIS nodes on top of which EIS components and workflows are deployed. Moreover, for every workflow specified in the EIS service profile, a source station, characterized by the corresponding statistical pattern, is generated. Formally, the following OCL [2] constraint gives the post condition of the first step of the generation procedure.

Measure	**Type**
mean-service-time	Real
mean-waiting-time	Real
mean-execution-time	Real
mean-system-throughput	Real
Stimuli	**Type**
statistical-pattern	Real |
	exp : Real -> Real |
	hexp : Real, Real -> Real |
	erlang : Real, Integer -> Real
Parameter	**Type**
thread-policy	Enum{single, multi, pool}
scheduling-policy	Enum{fifo, lifo, quantum, priority, order-preserving, sharing}
capacity	Integer | infinite
work-demands	Real |
	exp : Real -> Real |
	hexp : Real, Real -> Real |
	erlang : Real, Integer -> Real

Table .1. EIS performance measures, stimuli and parameters

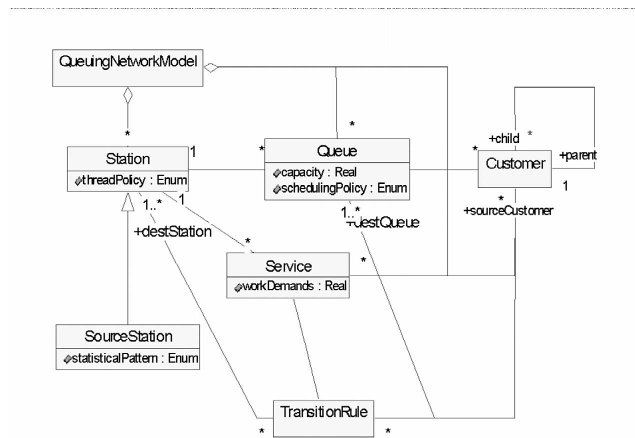


Figure 3. Basic meta-elements used for the specification of queuing networks.

```
EISSoftwareArchitecture:
  self.eisNode->forall(
   node |self.queuingNetworkModel.station->exists (
   st |st.name = node.name)) and
  self.eisServiceProfile.eisWorkflow->forall(
   wf |self.queuingNetworkModel.station->exists (
   st |st.name = wf.name->concat('SourceStation') and
   st.statistical-pattern = wf.statistical-pattern)
```

Then, for every EIS component, a queue is generated and associated with the appropriate station. Performance parameters related to the capacity and scheduling policy of the component are used to define the corresponding properties that characterize the queue. In addition, a service is generated for every EIS service provided by the component. The generated service is characterized by the work-demands required for the corresponding EIS service. Formally, the post condition of this step is:

```
EISSoftwareArchitecture:
  self.eisComponent->forall(
   res |self.queuingNetworkModel.station->select(
    st | st.name = res.eisNode.name).queue->exists(
    q | q.name = res.name and q.scheduling-policy =
      res.scheduling-policy and q.capacity = res.capacity
   ) and
   res.eisService->forall(
    eisserv |self.queuingNetworkModel.station->select(
     st | st.name = res.eisNode.name
    ).service->exists(
      serv | serv.name = eisserv.name and
       serv.work-demands = eisserv.work-demands)))
```

Technically, up to this point, the parsers for the Darwin and the workflow specification languages are used to parse the EIS architectural descriptions and to generate the queuing network stations. In the next step, for every workflow in the service profile and for every task `t` in this workflow, a queue, `tQueue`, is generated and associated with the corresponding station. The generated queue is used to synchronize the execution of tasks that depend on `t`. `tQueue` queues customers sent by tasks that depend on `t`, requesting `t`'s activation. Moreover, the service `tService` provided to customers queued in `tQueue` is generated and associated with the corresponding station. The post condition of this step is:

```
EISSoftwareArchitecture:
  self.eisServiceProfile.eisWorkflow->forall(
   wf | wf.eisTask->forall(
    t |self.queuingNetworkModel.station->select(
     st | st.name = t.eisNode.name).queue->exists(
     q | q.name = t.name->concat('Queue'))) and
   wf.eisTask->forall(
    t |self.queuingNetworkModel.station->select(
     st | st.name = t.eisNode.name
    ).service->exists(
     serv | serv.name = t.name->concat('Service'))))
```

The code of `tService` follows the pattern described below:

- The initiation of the workflow causes the creation of customers `initc` sent to the queue of each task `t`.
- Serving `initc` causes the generation of new sets of customers, one per alternative input set required by task `t`. Each new set of customers is sent to the stations that host queues of the tasks providing the corresponding outputs.
- `initc` waits until one of the new customer sets is served. Then, another set of customers is created and sent to the queues that correspond to the EIS components used by `t`. The exact code generated here depends on the way tasks use EIS services provided by EIS components.
- `initc` remains blocked until all of the created customers are served by the EIS components. Then, an output set is produced and customers waiting on stations for this particular output set are unblocked. Finally, customer `initc` is unblocked and destroyed.

Technically, to generate the queues and the services, used for the synchronization of tasks, we use the parsers for the Darwin and the workflow specification languages.

4.2. Example

Getting back to our example, the three components used by the tasks of the `BillingServiceWorkflow` are multi-threaded and are modeled to have an unlimited capacity. The policy according to which they serve requests is FIFO. Finally, the work demands for providing the EIS services associated with them are constant (we do not provide further details here due to the lack of space). Hence, for all three components we have:

```
EISComponent:
  self.thread-policy = multi and
  self.scheduling-policy = fifo and
  self.capacity = infinite
```

The `BillingServiceWorkflow` is initiated regularly at the end of each month. Hence we have:

```
BillingServiceWorkflow:
  self.statistical-pattern = 30*24*3600
```

A queuing network for QNAP2 is then generated simply, using the tool functionality, and according to the mapping defined in the previous subsection. In particular, the following elements are generated: 3 stations and the corresponding queues representing the components; 4 stations

```
1 /STATION/
2 NAME = PaymentQueue;
3 TYPE = INFINITE;
4 SERVICE =
5 BEGIN
6 IF(Payment-H(CUSTOMER.wfid, 4).STATE <> TRUE) THEN
7 BEGIN
8       SET(Payment-H(CUSTOMER.wfid, 4));
9       PRINT("Payment serving workflow", CUSTOMER.wfid);
10      tmp-Bill(1):= NEW(CUSTOMER);
11      tmp-Bill(1).wfid:= CUSTOMER.wfid;
12      tmp-Bill(1).Bill-IOS :=1;
13      tmp-Bill(1).all-avai:= NEW(FLAG);
14      TRANSIT(tmp-Bill(1), BillStation, Payment-CL);
15      WAITOR(tmp-Bill(1).all-avai);
16      TRANSIT(NEW(CUSTOMER), Department);
17      JOIN;
18      res_H := HISTOGR(s_pay, (0.33333334,0.33333334,0.33333334));
19      IF ((res_H >0.0) AND (res_H <= 0.33333334)) THEN
20      BEGIN
21         SET(pay_H(CUSTOMER.wfid, 1));
22      END
23      ELSE
24      IF ((res_H >0.33333334) AND (res_H <= 0.6666667)) THEN
25      BEGIN
26         SET(pay_H(CUSTOMER.wfid, 2));
27      END
28      ELSE
29         IF ((res_H >0.6666667) AND (res_H <= 1.0)) THEN
30         BEGIN
31            SET(pay_H(CUSTOMER.wfid, 3));
32         END;
33      TRANSIT(OUT);
 &.....................................
56      IF(OK-fin AND OK-Transfer AND OK-Payment AND OK-Bill AND OK-rec AND OK-Claim) THEN
57        SET(CUSTOMER.all-avai);
58 END; & of ELSE CLAUSE
59 END; & of service
```

Figure 4. Part of the code of the service provided to customers used for the synchronization of task `Payment` with task `Bill`.

hosting queues used for the synchronization of tasks; a source station whose statistical pattern equals to the one of the `BillingServiceWorkflow`.

Figure 4 gives part of the model of the station that hosts the queue used to synchronize the `Payment` task with the rest of the tasks of the `BillingServiceWorkflow` workflow. During the initiation of the workflow, an initiation customer is sent to the `PaymentQueue`. As shown in Figure 2, the `Payment` task depends on the completion of the `Bill` task. Consequently, serving the initiation customer results in the creation of a synchronization customer, which is sent to `BillQueue` requesting the initiation of the `Bill` task (Figure 4, lines 10-14). The initiation customer is blocked until the newly created synchronization customer is properly served (Figure 4, line 15). Then, a new customer is sent to the station that hosts `DepartmentQueue`, asking for a basic service. Once this customer is served, `Payment` completes and the initiation customer is destroyed (Figure 4, lines 16-33).

To give an idea of the complexity of the resulting model, its total size for the complex billing service workflow is 490 lines.

Measure	**Type**	
reliability	0..1	
Stimuli	**Properties**	
Failure	domain	Enum{time, value}
	perception	Enum{consistent, inconsistent}
Parameter	**Properties**	
Fault	nature	Enum{intentional, accidental}
	phase	Enum{design, operational}
	causes	Enum{physical, human}
	boundaries	Enum{internal, external}
	persistence	Enum{permanent,temporary}
	arrival-rate	Real

Table .2. EIS reliability measures, stimuli and parameters

5. Automated Reliability Analysis

The basic reliability measure for EIS is the probability that a workflow successfully completes during the lifetime of the EIS. Getting to the reliability parameters, EIS components, tasks and nodes may fail because of faults causing errors in their state. The manifestations of errors are failures [5]. Hence, faults are the basic parameters that affect the reliability of an EIS, while failures are the stimuli causing changes in the value of the reliability measure. Faults and failures are further characterized by properties given in Table .2. Different combinations of the values of those properties lead to the definition of fault and failure taxonomies (e.g. see [5]), facilitating the automated generation of traditional reliability models. Except for faults and errors, another parameter affecting reliability is design diversity. Frequently, more than one components provide similar services, which can be exploited towards achieving a particular objective. Such cases can be specified using the workflow specification language. In particular, EIS tasks may require one, or more alternative input-sets and may provide one, or more alternative output-sets. Hence, tasks representing N-Version-Parallel (NVP) and Recovery Block (RB) schemas [6] can be defined and taken into account for the generation of traditional reliability models.

5.1. Mapping EIS models on traditional reliability models

Reliability analysis techniques are typically based on state space models whose overall structure is given in Figure 5. A state space model consists

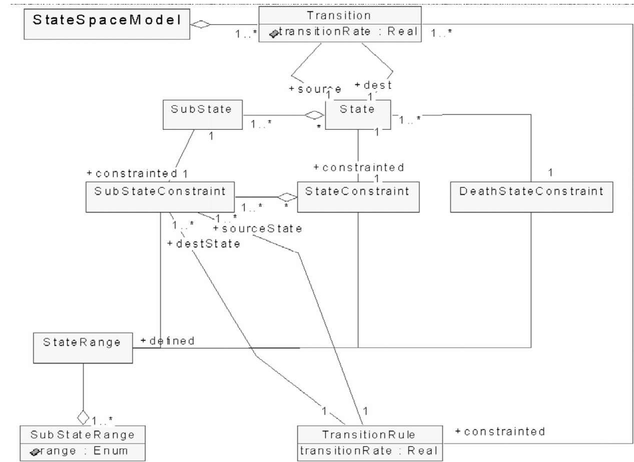


Figure 5. Basic meta-elements used for the specification of state space models.

of a set of transitions between states of the system. A state describes a situation where either the system operates correctly, or not. In the latter case the system is said to be in a *death state*. The state of the system depends on the state of its constituent elements. Hence, it can be seen as a composition of sub states, each one representing the situation of a constituent element. A state is constrained by the range of all possible situations that may occur. A state range can be modeled as a composition of sub state ranges, constraining the state of the elements that constitute the system. A transition is characterized by the rate by which the source situation changes into the target situation. If, for instance, the difference between the source and the target situation is the failure of a component, the transition rate equals to the failure rate of the component.

The specification of large state-space models is often too complex and error-prone. The approach proposed in [1] alleviates this problem. In particular, instead of specifying all possible state transitions, the authors propose specifying the state range of the system, a death state constraint, and transition rules between sets of states of the system. In a transition rule, the source and the target set of states are identified by constraints on the state range (e.g. if the system is in a state where more than 2 components are operational, then the system may get into a state where the number of components is reduced by one). Given the previous information, a complete state space model can be generated using the algorithm described in [1]. Briefly, the algorithm takes as input an initial state and recursively applies the set of the transition rules. During a

recursive step, the algorithm produces a transition to a state derived from the initial one. If the death state constraint holds for the resulting state, the recursion stops.

Based on the above, in the remainder we detail how to exploit the EIS architectural description to generate the information needed for the generation of a corresponding complete state space model. The first step towards that goal is to generate a state range definition for each workflow belonging to a given service profile. The state of a workflow is composed of the states of the tasks making up the workflow and the states of the nodes on top of which tasks and components are deployed.

The state of a task consists of a state representing the situation of the task itself and states representing the situations of the task's alternative input and output sets. The situation of a task depends on the kinds of faults that may cause the failure of this task. For instance, if the task fails due to permanent faults, its state may be `Waiting`, `Busy`, `Complete`, or `Failed`. If the task fails due to intermittent faults, its state may be `Waiting`, `Busy`, `Complete`, `FailedActive`, or `FailedPassive`.

The state of an input (resp. output) set, `ioset`, is composed of the states of the individual inputs `io` (resp. outputs) included in the set. If `io` is a notification, its state may be either `Available`, `NotYetAvailable`, or `NeverAvailable`; `io` is `NeverAvailable` if the task that provides it has failed, or completed by producing an output set that does not include `io`. If `io` is a reference to a component `c`, its state depends on the kind of faults that may cause the failure of `c`.

Based on the previous, the post condition of the generation of a state range is given below:

```
EISSoftwareArchitecture:
self.eisServiceProfile.eisWorkflow->forall(
 wf |wf.eisTask->forall(
  t | wf.stateRange.subStateRange->exists(
   str, strNode | strNode.name =
    t.eisNode.name->concat('StateRange') and
   str.name = t.name->concat('StateRange') and
   t.requires->union(t.provides)->forall(
     ioSet |str.stateRange.subStateRange->exists(
      str' |str'.name = ioSet.name->concat('StateRange') and
      ioSet->forall(
       io | str'.subStateRange->exists(
       str'', strNode'' | str''.name =
         io.name->concat('StateRange') and
        strNode''.name = io.eisNode.name->concat('StateRange')
      )))))))
```

After generating the state range definition for a workflow `wf`, the step that follows comprises the generation of transition rules for every task `t`

of `wf` and for the EIS nodes. Those rules depend on the kind of faults that may cause the failure of `t`. For permanent faults, the rules for task `t` follow the pattern below:

- If `wf` is in a state where `t` is `Waiting` then:
  - If an alternative input set `ioset` is available then `wf` may get into a state where `t` is `Busy`.
  - If none of the alternative input sets `ioset` may eventually become available then `wf` may get into a state where all tasks depending on `t` are aware about the fact that its output sets will never be available.

  The previous are, typically, fast transitions, i.e. the probability that they take place is close to 1.
- If `wf` is in a state where `t` is `Busy` due to the availability of `ioset` then `wf` may get into a state where:
  - `t` is `Complete`. Again, this is a fast transition.
  - `t` is `Failed` and all tasks depending on `t` are aware about the fact that its output sets will never be available. The rate of getting into this state equals to the arrival rate of the fault that caused the failure of `t`, i.e. `t.fault.arrival-rate`.
  - `t` is `Waiting` and `io` belonging to `ioset` is `Failed`. All EIS references `io'` used by other tasks of `wf`, for which `io'.eisComponent = io.eisComponent` holds, get into a `Failed` state. The rate of this transition equals to the arrival rate of the fault that caused the failure of `io`, i.e. `io.eisComponent.fault.arrival-rate`.

The rules for a node `n` are more obvious, and are not given here due to the lack of space. Finally, a death state constraint must be generated. In general, `wf` is in a death state if none of its output sets may eventually become available due to the unsuccessful termination of the tasks providing the corresponding outputs.

Technically, the generation of the information discussed in this section requires using the parsers for the Darwin and the workflow specification languages.

5.2. Example

Getting back to our example, from the workflow specification given in Figure 2 we can generate the necessary information that serves as input to the algorithm presented in [1]. More specifically, both the tasks of

the `BillingServiceWorkflow` and the components used by those tasks may fail due to permanent faults. Hence,

```
BillingServiceWorkflow:
  self.eisResource->forall(
   res | res.fault.persistence = permanent
  ) and
  self.eisServiceProfile.eisTask->forall(
   res | res.fault.persistence = permanent
  )
```

The state of the workflow is composed of the states of the `Bill`, `Payment`, `Transfer`, and `Claim` tasks, and the states of the nodes on top of which tasks and components are deployed. The range of each of those states is `Enum{Waiting, Busy, Complete, Failed}`.

Figure 6 gives the transition rules generated for the `Payment` task and used as input to the realization of the algorithm [1]. In particular, if the workflow is in a state where `Payment` is `Waiting` and its input set is available, then the workflow may get to a state where `Payment` is `Busy` (lines 1-4). If the workflow is in a state where `Payment` is in a `Busy` state, the workflow may get into a state where `Payment` is `Complete` (lines 18-29). Alternatively, the workflow may get into a state where `Payment` is `Failed` and the `Claim` and `Transfer` tasks are aware about the fact that the `Payment` outputs will never become available. (lines 30-37). The workflow reaches a death state if neither of its output sets may eventually become available.

The overall size of the model used as input for the algorithm [1], is 325 lines of code. Moreover, the generated Markov model contains 616 states. 282 out the 616 are death states. Finally, the model contains 2092 transitions.

6. Conclusion

In this paper, we presented an approach for automating the performance and reliability analysis of EIS systems. The approach is based on the formal definition of mappings between EIS architectural models and traditional performance and reliability models. The benefits of the proposed approach are both qualitative and quantitative. In particular, the quality of traditional performance and reliability models is assured since the required experience for building them is encapsulated in automated model generation procedures. Moreover, the cost of performing performance and reliability is minimized since the development of the corresponding traditional models is achieved automatically. It is worth-noticing that according to the authors of [2], 25% of the time required for performing architecture tradeoff analysis of software systems is actually

```
1 IF (Payment = WAITING) THEN
2   IF (PaymentInset00 = AVAILABLE AND
3       PaymentInset01 = AVAILABLE) THEN
4      TRANTO Payment = BUSY BY INPUT_AVAILABLE;
5   ELSE
6     IF (PaymentInset00 = NEVERAVAILABLE OR PaymentInset00 = FAILED OR
7  PaymentInset01 = NEVERAVAILABLE OR PaymentInset01 = FAILED) THEN
8      TRANTO
9      Payment = COMPLETE ,
10     PaymentOutset00 = NEVERAVAILABLE,
11     PaymentOutset10 = NEVERAVAILABLE,
12     TransferInset02 = NEVERAVAILABLE,
13     ClaimInset01 = NEVERAVAILABLE BY
14     INPUT_AVAILABLE;
15   ENDIF;
16   ENDIF;
17 ENDIF;
18 IF (Payment = BUSY) THEN
19  TRANTO Payment = COMPLETE,
20    PaymentOutset00 = AVAILABLE,
21    TransferInset02 = AVAILABLE BY
22    INPUT_AVAILABLE;
23 ENDIF;
24 IF (Payment = BUSY) THEN
25   TRANTO Payment = WAITING,
26      PaymentInset00 = FAILED BY
27      LAMBDA;
28 ENDIF;
29 IF (Payment = BUSY) THEN
30   TRANTO Payment = WAITING,
31      PaymentInset01 = FAILED BY
32      LAMBDA;
33 ENDIF;
34 IF (Payment = BUSY) THEN
35   TRANTO Payment = COMPLETE,
36      PaymentOutset10 = AVAILABLE,
37      ClaimInset01 = AVAILABLE BY
38      INPUT_AVAILABLE;
39 ENDIF;
40 IF (Payment = BUSY) THEN
41   TRANTO Payment = FAILED,
42    PaymentOutset00 = NEVERAVAILABLE,
43    PaymentOutset10 = NEVERAVAILABLE,
44    TransferInset02 = NEVERAVAILABLE,
45    ClaimInset01 = NEVERAVAILABLE BY
46    LAMBDA;
47 ENDIF;
```

Figure 6. Transition rules for the `Payment` task, used for the generation of a complete state space model for the `BillingServiceWorkflow`.

spent on building traditional quality models. The approach proposed in this paper enables decreasing this cost.

The approach presented here can be applied for automating the quality analysis of EISs regarding several other attributes. More specifically, the case of availability is pretty similar to the one of reliability. From our point of view, an interesting perspective is to extend this work towards the analysis of EISs regarding qualities attributes like openness and scalability.

Notes

1. www.simulog.com

2. OCL is a first order logic notation used for specifying constraints on UML models. OCL supports the basic logical operators (e.g. `and`, `or`, `forall`, `exists`, `implies`). Moreover, the `.` operator allows to navigate through associations defined in the UML model. For more details see [9]

References

[1] S. C. Johnson. Reliability Analysis of Large Complex Systems Using ASSIST. In *Proceedings of the 8th Digital Avionics Systems Conference*, pages 227–234. AIAA/IEEE, 1988.

[2] R. Kazman, S. J. Carriere, and S. G. Woods. Toward a discipline of scenario-based architectural engineering. *Annals of Software Engineering*, 9:5–33, 2000.

[3] M. Klein, R. Kazman, L. Bass, S. J. Carriereand M. Barbacci, and H. Lipson. Attribute-Based Architectural Styles. In *Proceedings of the First Working Conference on Software Architecture (WICSA1)*, pages 225–243. IFIP, Feb 1999.

[4] H. Kobayashi. *Modeling and Analysis : An Introduction to System Performance Evaluation Methodology*. Addison-Wesley, 1978.

[5] J-C. Laprie. Dependable Computing and Fault Tolerance : Concepts and Terminology. In *Proceedings of the 15th International Symposium on Fault-Tolerant Computing (FTCS-15)*, pages 2–11, 1985.

[6] J-C. Laprie, J. Arlat, C. Béounes, and K. Kanoun. Definition and analysis of hardware and software fault-tolerant architectures. *IEEE Computer*, 23(7):39–51, July 1990.

[7] H. Ludwig and K. Whittigham. Virtual Enterprise Co-ordinator - Agreement-Driven Gateways for Cross Organizational Workflow Management. In *Proceedings of the Interantional Joint Conference on Work Activities Coordination and Collaboration (WACC'99)*, Software Engineering Notes, pages 29–38. ACM, 1999.

[8] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying Distributed Software Architectures. In *Proceedings of the 5th European Software Engineering Conference (ESEC'95)*, number 989 in LNCS, pages 137–153. Springer Verlag, 1995.

[9] OMG. *Object Constraint Language Specification*, 1.1 edition, Sept 1997.

[10] OMG. UML Profile for Enterprise Distributed Object Computing. Technical report, OMG, 2000.

[11] S.M. Wheater, S.K. Shrivastava, and F. Ranno. A CORBA Compliant Transactional Workflow System for Internet Applications. In *Proceedings of MIDDLEWARE'98*, pages 3–18. IFIP, September 1998.