
Proceedings Paper:
Weyns, Danny and Calinescu, Radu orcid.org/0000-0002-2678-9260 (2015) Tele Assistance: : A Self-Adaptive Service-Based System Exemplar. In: 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). , pp. 88-92.

https://doi.org/10.1109/SEAMS.2015.27

Tele Assistance: A Self-Adaptive Service-Based System Examplar

Danny Weyns* and Radu Calinescu†
*Department of Computer Science at Linnaeus University, Växjö Campus, Sweden
Email: Danny.Weyns@lnu.se
†Department of Computer Science at the University of York, UK
Email: Radu.Calinescu@york.ac.uk

***Abstract*—Research on adaptive and self-managing systems is hindered by a lack of prototypical applications that researchers could use to evaluate and compare new methods, techniques and tools. To address this limitation, we introduce a reference implementation of a Tele Assistance System (TAS) for research on self-adaptation in the domain of service-based systems. Our TAS exemplar of service-based systems comes with pre-defined scenarios for comparing the effectiveness of different self-adaptation solutions. Other researchers can easily exploit the underlying service platform, reusable components and development method we devised for TAS to speed up the engineering of additional research exemplars for service-based systems.**

I. Introduction

Despite significant research over the past decade, the rigorous engineering of adaptive and self-managing systems remains a formidable challenge [7]. In similar circumstances affecting another area of software engineering, Feather *et al.* [9] proposed the use of requirements and specification exemplars to drive and communicate research advances, establish research agendas, and compare and contrast alternative approaches. Likewise, the research community suggested the use of exemplar systems as stepping stones to develop the necessary benchmarks, methods, techniques and tools for the engineering of self-adaptive software systems [5], [7]. However, only a few such exemplars have been proposed so far, with varying degrees of success. The most successful of these is the Znn.com news site exemplar, a web-based client-server system proposed by Carnegie Mellon researchers [6] and used, for instance, in [1], [12]. Another noteworthy example is the automated traffic routing problem proposed in [15], which focuses on decentralised adaption and its related challenges (e.g., scalability, robustness, and balancing the precision and performance of monitoring). However, so far, this exemplar has not been used actively.

Our paper contributes to this effort by proposing an exemplar for research on self-adaptation in the area of service-based systems (SBSs). SBSs are widely used in e-commerce, online banking, e-health and many other applications. In these systems, services offered by third-party providers are dynamically composed into workflows to deliver complex functionality. Most importantly for our purpose, SBSs increasingly rely on self-adaptation to cope with the uncertainties associated with third-party services, as the loose coupling of services makes online reconfiguration feasible.

We present a reference implementation of a Tele Assistance System (TAS) and a set of predefined scenarios for comparing self-adaptation solutions. TAS was originally introduced in [2], and has the advantage that it has already been used in the evaluation of several self-adaptation solutions [3], [4], [8], [10], albeit based on ad-hoc implementations, scenarios and evaluation metrics that make the comparison of these solutions and its use to evaluate other solutions very difficult. To address these limitations, we implemented TAS using our new Research Service Platform (ReSeP[1]), and we propose predefined concrete scenarios for its immediate use in the evaluation of self-adaptation solutions. In addition, other researchers can take advantage of ReSeP and its reusable components to speed up the engineering of new service-based system exemplars. We envisage that these contributions will help promote collaborative research and advance the engineering of adaptive and self-managing systems.

The paper is structured as follows. In Section II, we present the TAS exemplar and provide scenarios for comparing self-adaptation solutions. Section III explains the develop method we used for implementing the TAS with ReSeP. In Section IV, we illustrate how we used the TAS exemplar to compare two adaptation approaches for one of the scenarios. Finally, we summarise our conclusions in Section V.

II. SBS Exemplar and Adaptation Scenarios

Our service-based system exemplar is a Tele Assistance System (TAS) that provides health support to chronic condition sufferers within the comfort of their homes. TAS uses a combination of sensors embedded in a wearable device and remote services from healthcare, pharmacy and emergency service providers. As shown in Fig. 1, the TAS workflow takes periodical measurements of the vital parameters of a patient and employs a third-party medical service for their analysis. The analysis result may trigger the invocation of a pharmacy service to deliver new medication to the patient or to change his/her dose of medication, or the invocation of an alarm service leading, e.g., to an ambulance being dispatched to the patient. The same alarm service can be invoked directly by the patient, by using a panic button on the wearable device.

[1]Pronounce *re-ce-pe* like in 'recipe'.

TABLE I
GENERIC ADAPTATION SCENARIOS FOR SERVICE-BASED SYSTEMS

Scenario	Type of uncertainty [13]	Type of adaptation [2]–[4], [8], [10]	Type of requirements
S1	Unpredictable environment: service failure	Switch to equivalent service; Simultaneous invocation of several services for idempotent operation	QoS: Reliability, cost
S2	Unpredictable environment: variation of service response time	Switch to equivalent service; Simultaneous invocation of several services for idempotent operation	QoS: Performance, cost
S3	Incomplete information: new service	Use new service	QoS: Reliability, performance, cost
S4	Changing requirements: new goal	Change workflow architecture; Select new service	Functional: new operation
S5	Inadequate design: wrong operation sequence	Change workflow architecture	Functional: operation sequence compliance

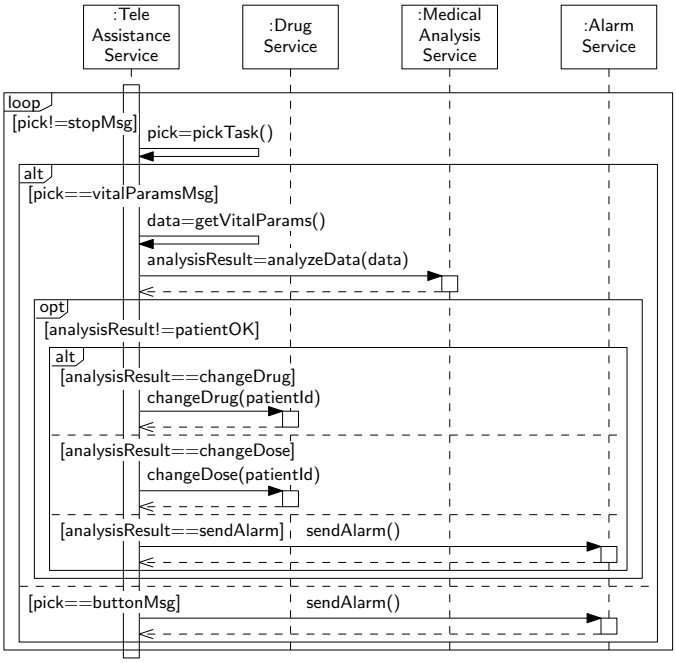

Fig. 1. TAS workflow

To enable the consistent use of TAS and future SBS exemplars for the evaluation, comparison and advance of self-adaptation solutions, we devised the generic adaptation scenarios from Table I. These scenarios are organised by type of uncertainty that makes self-adaptation necessary (cf. the taxonomy of uncertainty in [13]), type of adaptation required (cf. the SBS adaptations from, e.g., [2]–[4], [8], [10]), and type(s) of requirements that these adaptations aim to meet. Within these scenarios, we propose the evaluation and comparison of different self-adaptation solutions based on quality attributes and metrics described in [14] and summarised in Table II.

III. TAS IMPLEMENTATION WITH RESEP

A. The ReSeP Platform

Fig. 2 shows the main ReSeP components that reify the principles of Service-Oriented Architecture (SOA). We distinguish between atomic services, which offer functionality without depending on other services, and composite services, which represent compositions of atomic and other composite services. Service composition is specified by means of a workflow that is executed by a workflow engine. An example of a workflow specified with ReSeP's simple but expressive workflow specification language is available in Appendix A.

TABLE II
QUALITY ATTRIBUTES AND METRICS FOR THE EVALUATION AND COMPARISON OF SBS SELF-ADAPTATION SOLUTIONS

Quality attribute	Metrics
Reliability	Number of failed service invocations Number of specific operation sequence failures Mean time to recovery
Performance	Number of specific operation sequences exceeding allowed execution time
Cost	Cumulative service invocation cost over given time period
Functionality	Number of faulty process executions

For each available service, a service description stored in a service registry specifies its operations, unique address (i.e., *endpoint*) and custom properties such as cost and promised quality-of-service (QoS) attributes. A composite service can look up atomic services in the registry and maintains a local cache of available services. Service clients that invoke a composite service can provide a specification of the quality of service they require. The workflow uses this specification to select relevant services from the cache. E.g., a "high reliability" QoS requirement may lead to the selection of services with minimal (advertised) failure rate, and a "low cost" QoS requirement to the selection of minimal-cost services. Custom QoS requirements can be defined for new or combined quality attributes.

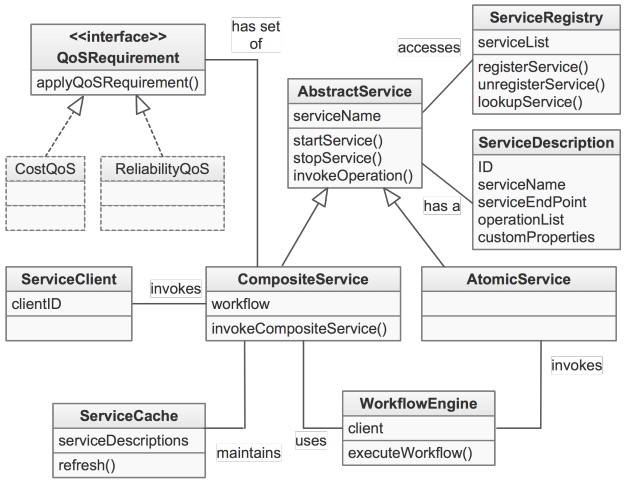

Fig. 2. ReSeP realisation of SOA principles

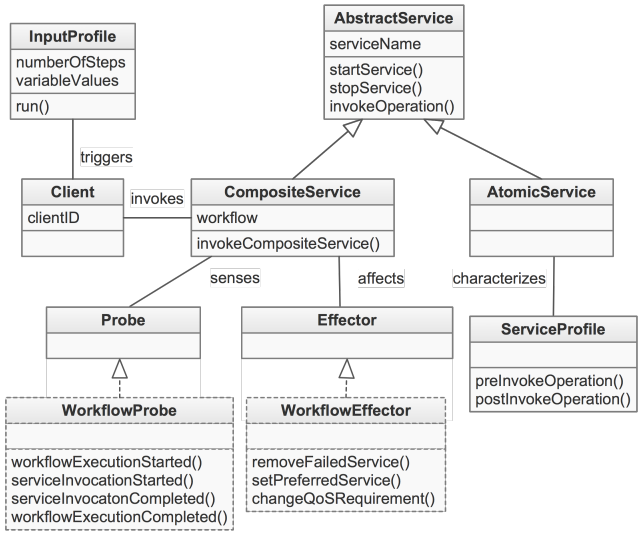

Fig. 3. Core experimentation features of ReSeP

The ReSeP components shown in Fig. 3 support the development of SBS exemplars. First, QoS behaviour of a service specified in *service profile* is "added" to the regular behaviour of the service operations, before and/or after their invocation. For example, a profile may specify the addition of a delay to the execution of a service invocation to model periods with different workloads. Second, ReSeP uses *input profiles* to define sequences of application-specific service invocations such as "*perform 200 invocations with a 30% probability of selecting one branch of the workflow, a 20% probability of selecting another branch, etc.*" Third, the platform offers *probes* that monitor the SBS, e.g., the WorkflowProbe from Fig. 3 monitors the start and completion of workflow executions and individual service invocations. Finally, ReSeP offers *effectors* that enable runtime manipulation of SBS architecture and parameters. An adaptation engine can use these probes and effectors to track SBS behaviour and adapt a SBS dynamically. ReSeP provides a set of pre-defined service and input profiles, probes and effectors; this set can be extended as needed.

B. TAS Realization with ReSeP

1) TAS Services: The drug, medical analysis and alarm services from Fig. 1 are realised as implementations of AtomicService, as illustrated by these AlarmService code snippets:

```
public class AlarmService extends AtomicService {
  public AlarmService(String id,String endpoint) {
     super(id, endpoint);
  }
  @ServiceOperation
  public boolean triggerAlarm(int patientID) {...}
}
...
AlarmService a1 = new AlarmService("Alarm1","queue1");
ServiceDescription sd = a1.getServiceDescription();
sd.getOp("triggerAlarm").setOpCost(...);
a1.register();
a1.startService();
```

Instantiating a service requires an ID and an endpoint (i.e., the name of a message queue for communication) and the configuration of its automatically created service description. The service is then registered and started. The only composite service from Fig. 1, TAS, is realised as shown below:

```
public class TAS extends CompositeService {
  public TAS(String id,String endpoint,String file){
    super(id, endpoint, file);
  }
  @ServiceOperation
  public boolean callTAS(String QoS,
                int patientID, int pick) {  ...  }
  ...
}
...
TAS as = new TAS("TASservice","tasq","TASworkflow");
as.addQoSRequirement("ReliabilityQoS",
    new ReliabilityQoS());
as.addQoSRequirement("CostQoS", new CostQoS());
client.invokeCompositeService("callTAS",
                          "CostQoS",patientID,pick);
```

The service is invoked with a required QoS requirement (from those pre-specified using its `addQoSRequirement` method), the patient ID, and the action to be performed (e.g., `pick==buttonMsg` to send an alarm).

2) Profiles, Probes and Effectors: Service profiles implemented as subclasses of `ServiceProfile` can override its `preInvokeOperation` and `postInvokeOperation` methods to emulate service failures, invocation delays, etc. Service profile instances can then be associated with services, e.g., the profile below introduces a 0.05 failure rate for the `AlarmService a1` defined earlier.

```
public class AlarmProfile extends ServiceProfile {
  private float failureRate = 0;
  public void setRate(float r) { failureRate = r; }
  @override
  public boolean preInvokeOperation() {
      return rand.nextFloat() > failureRate;
  }
}
...
AlarmProfile asp = new AlarmProfile();
asp.setRate(0.05);
a1.setServiceProfile(asp);
```

TAS provides a set of *input profiles* that specify and to execute particular sequences of invocations of the TAS service, each invocation being associated with predefined values for the workflow variables (types of actions, QoS requirements).

The TAS exemplar uses ReSeP probes to monitor service invocations and the cost of the invocations. TAS uses ReSeP effectors to dynamically update the parameters and architecture of its workflow. The code snippet below shows a workflow effector that offers different actions to adapt the TAS workflow.

```
public class WorkflowEffector extends Effector {
  ...
  public void removeFailedService
                          (ServiceDescription sd) {
    workflow.markUnavailable(sd);
  }
  public void setPreferredService
                          (ServiceDescription sd) {
    workflow.markAsPreferred(sd);
  }
  public void changeQoSRequirement(QoSRequirement r){
    workflow.setQoSRequirement(r);
  }
}
```

IV. Experimentation with TAS

Setting up a TAS experiment to evaluate and compare self-adaptation solutions is a six-step process, which we illustrate below with a concrete case study.

Step 1: Scenario and requirement selection—This step involves selecting one of the scenarios from Table I (e.g., 'S1: service failure'), and concrete requirements such as:

R1. *The percentage of TAS service invocations that fail to complete successfully is less than 1%*
R2. *The percentage of alarm invocations that do not complete successfully is less than 0.8%*
R3. Subject to R1 and R2 being satisfied, the cumulative cost of service invocations should be minimised.

Step 2: Service profile specification—A set of concrete services is assembled so that each TAS operation is supported by at least one service, and a service profile is specified for each service. Table III shows an example of a service set, with service profiles comprising a failure rate component like in the `AlarmProfile` from Section III-B2 and a cost component.

Step 3: Input profile definition—In this step, we define XML-encoded input profiles for the evaluation of self-adaptation solutions. For example, the input profile below specifies 500 invocation of the TAS service using the CostQoS requirement from Section III-B1. Of these invocations, 75% randomly chosen invocations will "pick" the medical analysis service and 25% the alarm service. All these parameters can be dynamically changed during the experiment.

```
<inputProfile>
   <maxSteps>500</maxSteps>
   <qosRequirement>CostQoS</qosRequirement>
   <variables>
      <variable>
         <name>pick</name>
         <values>
            <data>1</data> <ratio>0.75</ratio>
            <data>2</data> <ratio>0.25</ratio>
         </values>
      </variable>
      ...
</inputProfile>
```

Step 4: Evaluation metric definition—In this step, we define concrete metrics and add support for result visualisation. For our case study, we used the following evaluation metrics:

M1. *The percentage of alarm failures*
M2. *The percentage of failures of the assistance service*
M3. *The cumulative service invocation cost*

TABLE III
Concrete services with service profiles for TAS

Service Name	Failure Rate	Cost
Alarm Service 1	0.11	4.0
Alarm Service 2	0.04	12.0
Alarm Service 3	0.18	2.0
Medical Analysis Service 1	0.12	4.0
Medical Analysis Service 2	0.07	14.0
Medical Analysis Service 2	0.18	2.0
Drug Service 1	0.01	5.0

TABLE IV
Summary Results of the Case Study

Adaptation Strategy	Rates TAS Failures	Rates Alarm Failures	Cost (# Invocations)
No Adaptation	0.18	0.22	8.12K (1561)
Retry	0.005	0.01	9.95K (1981)
Select Reliable	0.0009	0.006	11.04K (1988)

TAS offers a graphical user interface with graphs and overview tables. For the case study we used predefined ReSeP graphs and tables, including a reliability graph that shows successful and failed service invocations per run of the input profile, and a cost graph of the cumulative cost for a run of the input profile. The overview tables summarize the results.

Step 5: Instrumentation—In this step, we select probes and effectors and connect the adaptation engine to be evaluated with TAS. If desired, additional probes and effectors may be implemented. For the case study, we tested two adaptation solutions realised with ActivFORMS [11], an adaptation engine that executes formally specified MAPE models. The two solutions implemented the following simple strategies:

Retry: *If a service fails, retry two times*
Select Reliable: *If a service fails, select the equivalent service with the lowest failure rate (and lowest cost if a tie)*

We used ReSeP probes and effectors including the `WorkflowProbe` (to track failed service invocations) and the `WorkflowEffector` (to adapt the workflow as needed).

Step 6: Execution—In the final step, we run the input profiles, collect data, and analyse the results. Table IV summarises the data collected for our case study using the earlier input profile, and Appendix A shows a reliability graph for the case study.

Both strategies realise R1, but only Select Reliable realises R2. Retry keeps the cost low but fails to satisfy the reliability for the alarm. Select Reliable's cost is 10% higher as Retry.

V. Conclusions

We presented TAS, a reference implementation of a service-based system (SBS) exemplar, and generic SBS adaptation scenarios associated with different types of uncertainty. TAS aims to serve the three key purposes of exemplars identified in [9]. First, it aims to promote research and understanding among multiple researchers and research groups, through enabling the comparison of different self-adaptation approaches, without favouring any particular approach. Second, TAS aims to serve the advance of single research efforts by reducing the time required to evaluate self-adaptation solutions. Finally, it aims to contribute to advancing the practice of engineering self-adaptive systems, by being a realistic example of a widely used type of software system. We therefore hope that the research community will use the TAS exemplar – and the underlying ReSeP service platform – to evaluate and compare research advances in adaptive and self-managing systems, and drive their further development.

The exemplar is available via the SEAMS exemplar website: http://self-adaptive.org/exemplars/tas.

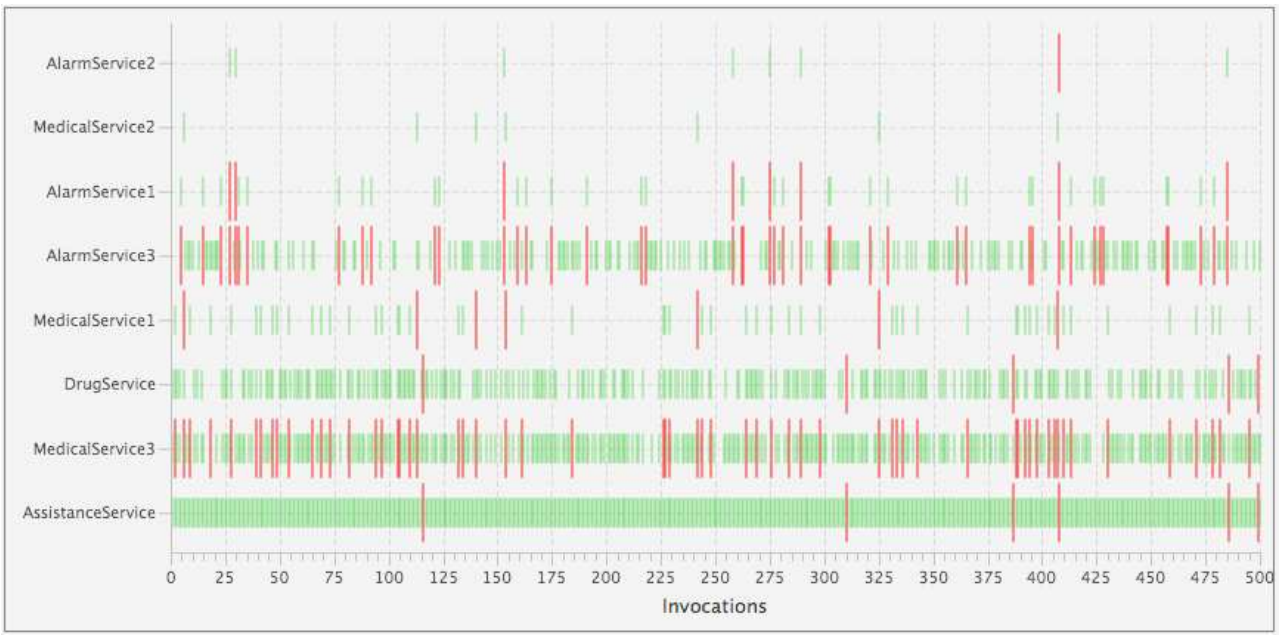

Fig. 4. Reliability graph for the case study; successful service invocations are represented by small vertical lines and failed invocations by larger lines.

ACKNOWLEDGMENT

The authors would like to thank M. Usman Iftikhar and Yifan Ruan for the implementation of TAS with ReSeP.

REFERENCES

[1] K. Angelopoulos, V. Souza, and J. Pimentel. Requirements and architectural approaches to adaptive software systems: A comparative study. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS'13, 2013.

[2] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of web service compositions. *Software, IET*, 1(6):219–232, 2007.

[3] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, September 2012.

[4] R. Calinescu, Lars Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS management and optimization in service-based systems. *Software Engineering, IEEE Transactions on*, 37(3):387–409, May 2011.

[5] B. Cheng et al. Software engineering for self-adaptive systems: A research road map. Lecture Notes in Computer Science vol. 5525. Springer, 2009.

[6] S.-W. Cheng, D. Garlan, and B. Schmerl. Evaluating the effectiveness of the rainbow self-adaptive system. In *Software Engineering for Adaptive and Self-Managing Systems*, 2009.

[7] R. de Lemos et al. Software engineering for self-adaptive systems: A second research roadmap. Lecture Notes in Computer Science vol. 7475. Springer, 2013.

[8] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *International Conference on Software Engineering, ICSE'09*, 2009.

[9] M. Feather, S. Fickas, A. Finkelstein, and A. van Lamsweerde. Requirements and specification exemplars. *Automated Software Engineering*, 4(4):419–438, 1997.

[10] A. Filieri, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Conquering complexity via seamless integration of design-time and run-time verification. In *Conquering Complexity*. Springer, 2012.

[11] M. U. Iftikhar and D. Weyns. Activforms: Active formal models for self-adaptation. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS'14, 2014.

[12] M. Luckey, B. Nagel, C. Gerth, and G. Engels. Adapt cases: Extending use cases for adaptive systems. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2011.

[13] A.J. Ramirez, A.C. Jensen, and B.H.C. Cheng. A taxonomy of uncertainty for dynamically adaptive systems. In *Software Engineering for Adaptive and Self-Managing Systems, SEAMS'12*, 2012.

[14] N. Villegas, H. Müller, G. Tamura, L. Duchien, and R. Casallas. A framework for evaluating quality-driven self-adaptive software systems. In *Software Engineering for Adaptive and Self-Managing Systems*, 2011.

[15] J. Wuttke, Y. Brun, A. Gorla, and J. Ramaswamy. Traffic routing for evaluating self-adaptation. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2012.

APPENDIX A

Reliability Graph

Fig. 4 shows a reliability graph for a run with the CostQoS input profile where ActivFORMS with the Reliable Selection strategy is used for adaptation. Notice for instance the call of the alarm at invocation number 153: the AlarmService2 deals with a failure of AlarmService1 that itself was not able to manage a failure of AlarmService3.

TAS Workflow

The code below shows the TAS workflow specified with the ReSeP workflow specification language.

```
START [patientId,pick]
     if (pick == vitalParamsMsg) {
         data = this.getVitalParameters()
         analysisResult =
            MedicalAnalysisService.analyzeData(data)
         if (analysisResult == changeDrug)
            DrugService.changeDrug(patientId)
        else if (analysisResult == changeDoses)
            DrugService.changeDoses(patientId)
        else if (analysisResult == sendAlarm)
            AlarmService.triggerAlarm(patientId)
     }
     else if (pick == pButtonMsg) {
         AlarmService.triggerAlarm(patientId)
}
RETURN
```