ExCovery – A Framework for Distributed System Experiments and a Case Study of Service Discovery

Andreas Dittrich∗, Stefan Wanja† and Miroslaw Malek∗

∗ Advanced Learning and Research Institute (ALaRI), Università della Svizzera italiana, Lugano, Switzerland
Email: {andreas.dittrich,miroslaw.malek}@usi.ch
† DSPECIALISTS Digitale Audio- und Messsysteme GmbH, Berlin, Germany
Email: stefan.wanja@dspecialists.de

Abstract—Experiments are a fundamental part of science. They are needed when the system under evaluation is too complex to be analytically described and they serve to empirically validate hypotheses. This work presents the experimentation framework ExCovery for dependability analysis of distributed processes. It provides concepts that cover the description, execution, measurement and storage of experiments. These concepts foster transparency and repeatability of experiments for further sharing and comparison. ExCovery has been tried and refined in a manifold of dependability related experiments during the last two years. A case study is provided to describe service discovery (SD) as experiment process (EP). A working prototype for IP networks runs on the Distributed Embedded System (DES) wireless testbed at the Freie Universität Berlin.

Keywords—Distributed Systems; Experiment; Experiment Framework; Tool Description

I. INTRODUCTION

Experiments are a fundamental part of science, needed when the system under evaluation is too complex to be analytically described and serving to empirically validate hypotheses. Experiments play an important role in computer science by supporting theories inferred from observations or mathematical models. With increasing complexity of computer systems and networks, exploratory experiments are themselves the source of such theories. However, due to their diverse focuses it remains difficult to repeat, classify, evaluate and compare the different experiment results. A consistent experimentation environment (EE) could help to unify related experiments and thus, greatly improve the impact of individual results. In this work, we present ExCovery, an EE for dependability research of distributed processes. A formal description to specify experiments has been developed, which forms the basis of ExCovery. It allows for automatic checking, execution and additional features, such as visualisation of experiments. ExCovery is expected to foster repeatability and transparency by offering a unified experiment description, measurement mechanism and storage of results.

Service oriented architecture (SOA) enforces the principle of discoverability, which means that structured data is added to service descriptions to effectively publish and discover individual providers. Communication of this data is done using service discovery (SD). In a case study, we show how to use ExCovery for experiments on SD. As a time-critical operation, one key property of SD is responsiveness – the probability that a number of SMs is found within a deadline, as required by the application calling SD. ExCovery was originally developed to support and validate research on SD responsiveness, as in [1], [2]. Due to space constraints, this work covers the abstract description of these experiments, their results will be published in future work. It should be noted that code listings presented in this paper have been shortened for illustrative purposes. The full code and descriptions are available on request and a repository will shortly be publicly accessible.

The rest of the paper is structured as follows. Section II covers the topics of scientific experimentation and design of experiments. Section III contains background information about service discovery. ExCovery is presented in Section IV its concepts illustrated with exemplary experiment description code. The description of SD as a specific experiment process follows in Section V. An overview of the current ExCovery prototype implementation is given in Section VI. Section VII concludes the work.

II. THE ART OF EXPERIMENTATION

The subject of an experiment can be characterized as a black box process as illustrated in Figure 1. Inputs or factors can be controlled, outputs or responses observed. During an experiment it needs to be identified which factors exist and how they influence the responses. This can be done one factor at a time or by manipulating multiple factors in a factorial experiment. Usually, experiments are run in
series to capture the variation among multiple runs of the same experiment. Such series of controlled experiments are called experimental system [3]. Experiments need to be reliable in a sense that they must be verifiable when repeated under similar conditions. Experiment design must therefore keep repetition in mind and its publication must contain all necessary information to do so. Experiments further need to fulfill requirements for internal and external validity. Internal validity means that the causal relationships between the factors and responses should be verified. External validity deals with the generalization of experiment results.

A. Basic Terms and Principles

In the context of this work, an EE is defined as a set of tools with the purpose of describing, executing and evaluating experiments on a given subject, using a methodology specific to that subject. The actual setting in which the experiments and the EE are run is called platform.

1) Experiment Factors: Factors are the different sources of variation among individual experiments. A treatment factor is “any substance or item whose effect on the data is to be studied” [4]. It can have continuous values but usually has discrete levels that are to be applied to study its effect. Treatment factors can be further classified as design factors, intentionally varied during the experiment, held-constant factors, whose impact is intentionally neglected and allowed-to-vary factors, known to have a minor influence that can be compensated by applying randomization and replication (see Section II-A3). Nuisance factors on the other hand have an unwanted effect on the output. They can be divided into controllable and uncontrollable nuisance factors. The former, often called blocking factors, can be fixed by the experimenter to reduce their impact on the response. The latter, also called covariates, can not be set but measured. Their effects can be minimized by covariance analysis. A noise factor causes random variations in the responses.

2) Experiment Design: A treatment is the entire description of what can be applied to the treatment factors of an experimental unit (EU), the smallest unit to which such treatment can be applied [6]. An observational unit (OU) then is the smallest unit on which a response will be measured. Experiment design defines which treatments are to be observed on which EUs [4]. Following [7], it can be divided into treatment design, a specification of the treatments used in an experiment, error control design, defining how specified treatments are to be applied to reduce unwanted variations, and sampling and observation design, which decides on the OUs and whether univariate or multivariate observations are to be taken. Proper experiment design strives to maximize the gained information per run by optimized structuring of factors and factor level variations over a required number of runs.

3) Experiment Validity: To improve the validity of experiments, three interconnected principles are applied. Replication increases the number of experiment runs to be able to average out random errors and to collect data about the variation in responses. Blocking means partitioning observations into groups, such that observations in each group are collected under similar conditions [4]. Statistical analysis requires that observations are independently distributed random variables [5]. Randomizing the assignment of treatments to EUs as well as the temporal order and spacial choice of multiple runs takes care of this requirement. It generally strengthens the external validity if an experiment is run in a diversity of platforms (see Section II-B1).

B. Experimentation Environment

In general, an EE allows to perform a certain class of experiments in a controlled environment. It facilitates the identification and manipulation of factors and the observation of these manipulations on the responses. For covariates, the EE should allow to record them to be considered during analysis at a later stage. To foster the repeatability, correctness and transparency of experiments, an EE should use a description for setup, execution and evaluation of experiments. A common output format for measurements, logs and diverse meta information should be provided.

1) Network Experimentation Platforms: ExCovery focuses on experiments to evaluate the dependability of distributed processes, e.g. network protocols. Thus, it relies on network experimentation platforms, such as simulators and testbeds. Simulators are software artifacts that simulate real-world processes by acting according to an abstract model. They can be discrete event-driven simulators, which calculate the state of the simulated object only when its state changes, or real-time simulators, which calculate the continuous behavior of the simulated object over time. There also exist mixed forms. While simulators have a perfect reproducibility of experiments, good scalability and generally a reduced execution time, their abstractions often struggle to capture the properties and behavior of real-world distributed systems [8]. A testbed is made of real network devices. They allow less control over factors but measurements are the result of a realistic interplay of factors. As such, testbeds usually allow to represent a specific environment (e.g. wireless mesh or large scale internet network) very well. An approach to unify generic network experimentation across simulators, emulators and testbeds is proposed with NEPI [9], an integration framework for network experimentation.

Another approach focused on testbeds is Weevil [10], [11].

III. Service Discovery Fundamentals

Among the principles introduced by SOA to support its paradigm is discoverability. Structured data is added to services to be effectively published, discovered and interpreted. Service discovery protocols (SDPs) take care of communicating this data, to announce, enumerate and sort existing service instances.
A. Basic Discovery Concepts and Roles

An abstract service class is provided by concrete service instances in the network. A set of service classes S can be provided on a set of providers P which then use an SD protocol to make the service known to interested users C, possibly supported by a a set of service registries R. For the remainder of this paper, we will use the taxonomy of a general SD model developed by Dabrowski et al. [12], in which these roles are called service manager (SM), service user (SU) and service cache manager (SCM). An SM publishes its service on behalf of a service provider either autonomously or via an SCM. It makes a service description available with information on how and where its service can be invoked. The SU discovers services on behalf of a user either by passively listening to announcements done by SAs or SCMs, by actively sending out queries to look for them, or by doing both. Discovery can happen in separate steps, enumerating discoverable instances first and then selectively retrieving the description. Also, not only services can be discovered, but administrative scopes, SCMs and service types, depending on the SDP. Finally, an SCM caches service descriptions of multiple SMs to maintain a list of present services that can be queried by SUs. SCMs are usually used to improve scalability.

B. Discovery Architecture and Communication

Two different SD architectures can be distinguished, as depicted in Figure 2. In two-party or decentralized architecture, there exist only SUs and SMs in the network which communicate directly among each other. The architecture is called three-party or centralized if there is one or more SCM present. There exist mixed forms that can switch among two- and three-party, called adaptive or hybrid architectures.

Depending on role and architecture, different communication types are used, unicast, multicast or broadcast. Furthermore, the communication scheme used for discovery can be classified as passive (or lazy), active (or aggressive) or directed. In passive discovery, SUs discover discoverable items only by listening to their unsolicited announcements. When doing active discovery, SUs actively send out multi- or broadcast queries. In directed discovery, SUs actively send unicast queries to a given SCM or SM. There are many messages used by the SDPs to coordinate the distributed system, maintain a consistent state and optimize network traffic. The currently most common SDPs are presented and compared in [12]–[14]. In [15]–[17], SDPs for pervasive and ubiquitous computing systems are surveyed, which are the target platform of the prototype in Section VI.

IV. THE EXPERIMENTATION ENVIRONMENT ExCovery

We will now present the main concepts of the proposed experimentation environment ExCovery with its core, the formal abstract description of an experiment using the extensible markup language (XML). An XML schema is provided with the framework code. It includes definitions of the experiment with its input factors, the process to be examined, of fault injections or manipulations and diverse platform specific and informative declarations. ExCovery further provides a unified measurement concept that determines which and how data are stored for later analysis.

An overview of the different concepts and the experiment work flow is illustrated in Figure 3. In the first preparation step, the experiment is designed by the experimenter following guidelines as mentioned in Section II-A2. The individual descriptions are explained in Section IV-C. Second, platform setup is necessary to prepare the translation of descriptions to the target platform. This could include the deployment of executables and configuration files. The experiment is then executed by the experiment master as specified in the description. Each run is a sequence of actions performed on the participating nodes, described as the main process under evaluation and a set of injected faults or manipulations. The master and all nodes monitor and record dedicated parameters during each run, such as raw packet captures and the complete temporal sequence of actions and events. These data will be saved in a temporary location locally. After experiment execution, the collected data are collected and conditioned so that a common time base for all actions, events and packet measurements is established. Finally, data are stored into a single results database that contains all conditioned measurement data, created log files and the complete experiment plan with the exact sequence of treatments (see Sections IV-B and IV-F).

A. Platform Requirements

To integrate a specific target platform in ExCovery, it must support several features. Most of the features are needed to establish a controllable environment or to compensate for missing control and to allow detailed measurements. As such, these are mainly an issues for testbeds, simulators generally can be integrated with less effort.

1) Experiment Management: There must be a separate, non-interfering and reliable communication channel between the experiment master and the nodes participating in the experiment process (EP). During experiments, full and privileged access to all nodes is mandatory. The platform needs to cleanly separate concerns of multiple users.
2) Connection Control: Full control over the network connections of the individual network nodes is needed. Network interfaces need to support activation and deactivation. Furthermore, it needs to be possible to manipulate packets sent over these interfaces based on defined rules, such as dropping, delaying, reordering, and modifying their content.

3) Measurement: There must be methods to capture packets with their exact local timestamps and their complete and unaltered content. To facilitate a comprehensive subsequent analysis, a packet tracking mechanism is required. In testbeds, this means tracking the routes of packets hop by hop, or attaching unique identifiers to packets [18]. Finally, the platform needs to support time synchronization among all participating nodes and quantification of the synchronization error.

B. Measurement and Recording

This section clarifies the basic observations that are possible using ExCovery, how they can be observed and how this can help to unify related experiments. ExCovery follows the principle of collecting as much data as possible to support diverse analyses on the same experiment data at later time, emphasizing reusability and repeatability. Basic recordable data are the results of protocol operations as reflected by state changes on the participants and network messages sent among participants. ExCovery supports a plugin concept to extend these data with custom measurements.

1) Events: State changes on nodes in the context of ExCovery reflect events and occur, for example, when an experiment run is initialized or when a fault injection is started. Events are associated with the node on which they occur. They contain a local time stamp and may have additional parameters, such as the identifier of the initialized experiment run. To control the experiment execution, nodes can be synchronized using global events (see Section IV-C2).

2) Packets: Packets are the basic communication data of network protocols. As opposed to events, single packets are not easily identified: Their location changes as they traverse the network, retransmissions and network loops complicate the correct localization at any given time. Packets are recorded to facilitate verification of the recorded event list and to derive statistical connection parameters. A measured packet consists of a time stamp, representing the local occurrence of that packet, a unique identifier, a source and destination network address and the packet content itself.

3) Time: Events and packets have local time stamps of the node they were measured on. ExCovery defines mandatory measurements to be done before each run to estimate the time difference of each participant to a reference clock. This allows to construct a valid global time line of events and packets, avoiding causal conflicts due to local clocks deviating between experiment runs.

4) Topology: To improve repeatability, a rudimentary description of the network topology is measured as hop count between the participating nodes. This is is done before and after executing an experiment. A more advanced topology recording is anticipated for future versions of ExCovery.

5) Recording: Each node has its own temporary storage for recorded data, organized into data belonging to single runs and to the entire experiment. Synchronization measurements are stored on the experiment master. Plugins have a separate storage location on the node where the custom measurements are done. ExCovery does not impose a specific storage mechanism but requires that it is accessible during the subsequent collection and conditioning phase (see Section IV-F).

C. Abstract Experiment Description and Execution

ExCovery executes experiments on the base of an abstract description made up of three parts. The first contains the experiment design, which factors are applied in which combination and order. The second contains manipulations on the process environment and the participants themselves, detailed in Section IV-D. The third describes the distributed process to be examined. In the following, the individual elements of the abstract description are explained.

Factor Part of the treatment applied to the EU. Consists of a set of levels. Depending on the design, levels are applied one after another (OFAT) or randomized.

List of factors Contains all factors used, sorted. In an OFAT design the first factor varies least often during execution while the last factor changes every run.

Level Concrete value a factor can take, as input variables to the sub-processes of each run. Levels can be of different types. As such, they can control type and duration
of fault injections (see Section IV-D) or represent mappings of abstract nodes to actors.

**Set of levels** All levels that should be applied during the experiment. Order of application is determined by the factor definition.

**Replication factor** Parameter defining an integer number of replications to be done with each treatment.

**Abstract node** Actor of the EP or of a node specific fault injector. Identified by a node identifier, such as a unique host name.

**Environment node** A node not participating as actor in any node specific process. Used e.g. to generate load.

**Actor description** Process prototype to be executed on one specific actor of the EP. Each abstract node is mapped to one actor description, multiple abstract nodes can instantiate the same actor description.

**Experiment process** Experiment operation that is to be executed and measured. Consists of actions performed on multiple nodes, synchronized by flow control functions that wait for a certain time or for certain events.

**Manipulation process** Main part of the treatment. Similar to EPs, represents a sequence of faults or impairments that should happen on a node.

**Environment manipulation process** Like experiment and manipulation processes but not node specific. Controls environment manipulations such as traffic generation.

Figure 4 shows a rudimentary beginning of an experiment description. Two abstract nodes A and B are to be mapped by the processes described later. For basic experiment classification, two parameters describing the discovery architecture and protocol are defined as key-value pairs.

1) **Execution**:

To execute the overall experiment and its individual runs from the abstract description, ExCovery generates treatment plans from replications, the factors and their levels. Plans are OFAT if no custom factor level variation plan is given. The various random values used in ExCovery are generated using pseudo-random generators. This allows for perfect repeatability of random sequences when initialized with the same seed. Which seed is used for initialization is clearly defined in the experiment description so that all random sequences can be reproduced.

ExCovery uses four internal functions for the experiment flow. Experiments are initialized by calling experiment_init on every participant, which takes care of the necessary preparations before all individual experiment runs. Each run is then initialized by run_init. There further exist the respective exit functions run_exit and experiment_exit.

Figure 5 shows the definition of several factors and their levels. First, the abstract nodes defined in Figure 4 are assigned actor roles actr0 and actrl. Then, two different factors are defined with various levels to describe the load generation that is to be applied during experiments, in this case a random number of 5 and 20 node pairs that will exchange data with first 10, 50 and then 100 kilobits per second. Each treatment will be repeated 1000 times.

Each run consists of the three phases preparation, execution and clean-up. During preparation, the whole environment of the EP must be reset to a defined initial working condition. Software agents are initialized. In testbeds, for example, network packets generated in previous runs must be dropped on all participants. Preliminary measurements can be done to compensate for incomplete control over the environment, such as clock offsets for all participants. During execution, the actual EP is executed, observed and recorded. Clean-up takes care of correctly terminating a run on each participant.

2) **Description of Processes**: ExCovery provides common mechanisms to control execution of the two types of defined processes. Abstract node processes are mapped to real nodes during experiment execution, such as protocol actions or fault injection processes. Environment processes are performed by all nodes, such as dropping packets on all network interfaces to reset the environment. Every process is described as a sequence of actions. Processes run concurrently so one needs to consider timing and desired or necessary dependencies. ExCovery defines methods for synchronization to provide basic flow control.
wait_for_time Process waits for a fixed delay in seconds.
wait_for_event Process waits until the specified event is registered on any participant. An event can be specified by its name, location and any of its parameters. The location is either a single abstract node or a subset of nodes specified by an actor role. Event parameters can be of diverse types. If omitted, they default to "any". A time-out in seconds can be set.
wait_marker Creates a time stamp used by the next wait_for_event call, which considers only events occurring after that time stamp.
event_flag Used to create local events to let process actions depend directly on each other.

Besides these flow control functions, there are process specific actions, environment actions and manipulation actions. Each action can have a list of parameters, allowing the description of manifold scenarios. In Section IV-C3 SD as an EP is described to illustrate this. Manipulation processes are described in Section IV-D3 Figure 6 shows a code fragment where the different processes are defined, without the actual sequences of actions that will be described later. Among the node processes, the role actr0 is defined and as possible actor nodes, the abstract nodes fact_nodes from the factor list are referenced. Environment processes do not need a definition of nodes.

D. Manipulation Processes

ExCovery has a concept for intentional manipulations done on participant nodes and on their network environment. Manipulations cover direct fault injections that cause failures in a targeted area. Fault provocation is used when direct injection is not desirable or possible and characterizes actions that are known to provoke failures in a targeted area. The main faults considered are communication faults. ExCovery provides a simplified fault model to allow for the description of basic fault behavior with a set of common parameters. The duration specifies the amount of time a fault should be applied to the target. The rate specifies a percentage of a given duration in which a fault is active. The fault is active in one continuous block, its activation time is chosen randomly using the randomseed.

1) Fault Injections: Mechanisms for fault injection are explained in the following. In addition to the common parameters, injections can have custom parameters to further define their behavior. It should be noted that all injected faults add up to already existing communication faults in the target platform. Also, whenever the term packet is used it refers to packets belonging to the EP.
Interface fault No messages are transmitted or received on the specified interface in the specified direction as long as this fault is active. Direction can be receive, transmit, both, or chosen randomly.
Message loss Defines probability for every packet to be dropped. Direction is analogous to the interface fault.

Figure 6. Template for the description of node and environment processes.

Message delay Applies a constant delay to every packet.
Path loss, path delay Path loss and delay are message loss and delay faults, selectively affecting only the communication between the target and a given second node.

2) Environment Manipulations: Environment manipulations are applied on a global level and involve possibly all specified environment nodes. Manipulations include the previously defined fault injections. Additionally, the following manipulations can be applied.
Traffic generator Creates network load between a given number of node pairs. Each pair bidirectionally communicates at a given data rate (see also Figure 5). Pairs can be randomly chosen from the acting nodes, non-acting nodes or all nodes. Pairs vary between runs as determined by a switch amount parameter.
Drop all packets All experiment nodes stop receiving, sending and forwarding EP packets.

Every fault injection and environment manipulation but the traffic generator is started only once and without a given duration, needs to be explicitly stopped. Given is just the default list supported by ExCovery. ExCovery provides also a generic function with an arbitrary list of parameters that are given to the acting nodes to be executed. However, an experimenter should preferably extend ExCovery by defining a plugin with new functions and their implementation.

3) Description of Manipulation Processes: Manipulation processes are defined in the experiment description as a series of actions and events. This list is executed sequentially and can contain flow control functions as described in Section IV-C2. A node manipulation process is created for each abstract node it is specified for while the environment manipulation process is implicitly supported on all nodes. The specific actions activate or deactivate the faults and manipulations as detailed in Section IV-D1 and IV-D2. One event is generated by each action to signal its start or stop, respectively. Parameters of these actions can be constant or varied during experiment execution. Variation is realized by references to factors instead of fixed values.

The manipulation actions can be used to extend the EP description or to define separate manipulation processes. This
depends on whether manipulations shall be synchronous with the EP or autonomous. Figure 7 shows a shortened listing of a traffic generation process. After generating a ready_to_init event, it uses the factors from Figure 5 to choose and configure traffic generation by a set of environment nodes, switching one pair of nodes in every run. The manipulation remains active until done is registered.

E. Description of Specifics

To instantiate an abstract experiment description on a concrete platform several specific settings are necessary. For reasons of brevity, this section explains only the most important settings included in the experiment description. First, a mapping of abstract and environment nodes to concrete usable nodes of the platform is required for execution. This mapping can change among experiments due to the unavailability of nodes or when deliberately changing them. ExCovery identifies nodes by their host name and IP address. The host name should be constant during an experiment run. When an IP address changes due to reconfiguration of a network interface, for example after a injection of such a fault, an event is generated to signal this. Second, an experimenter can define a list of special parameters in the description file that expose specific parameters used in the implementation. This allows modifications on ExCovery’s execution programs to be reused for many experiments without having to modify the implementation each time.

Figure 8 illustrates a compact version of a platform specification. Two actor nodes and three environment nodes exist. Actor nodes map to an abstract node id that has been previously defined (see Figure 4). All nodes have a unique identifier and a network address that can later be used to analyze the recorded event and packet lists.

F. Measurement Storage and Conditioning

ExCovery provides three levels of storage for experiments, with defined data structures. This allows reusable data access functions among experiments. The first storage level is the abstract experiment description itself, stored in an XML document. The second level is an intermediate storage for all concrete experiment data: experiment results and the software artifacts used during execution. Each log file and measurement is stored corresponding to a run identifier and a network address that can later be used to analyze the recorded event and packet lists.

Table I shows a subset of the tables and their attributes on the third level. The table ExperimentInfo represents the experiment as a whole and contains only one tuple made of the abstract experiment description, the version of ExCovery and a descriptive name and comment. Logs contains all raw log files and EEFiles the used ExCovery executables. In ExperimentMeasurements, specific named measurements are stored that are done once per experiment. As for run based data, RunInfos contains for each run and node the start time of the run and the offset of the node clock to the reference clock. Custom measurements are stored in ExtraRunMeasurements. The table Packets contains for each packet the common time stamp of detection, its originating node and the raw packet data. The Events table lists all recorded events and their parameters, identified by the run, the originating node and a common time stamp. This schema represents a preliminary approach to store data. Several future improvements are possible, for example by using a dimensional database model to store experiments in a data warehouse structure.
V. Abstract Service Discovery Processes Description

We will now explicate how to describe generic service discovery (SD) as an EP to be used within ExCovery. It provides a temporal and causal sequence of actions on the participating nodes as introduced in Section III, facilitating flow control functions from Section IV-C2. The description can contain actors for SM, SU, or SCM. For each actor a number of instances can be created to represent all participants of the SD process.

The model developed in [12] defines a set of main operations for a generic SD process, namely “Configuration Discovery and Monitoring”, “Registrations and Extension”, “Service-Description Discovery and Monitoring”, and “Variable Discovery and Monitoring”. These will be considered in the abstract SD process description, with an optional list of parameters to define specific variants. The details of executing the description are left to the SD protocol (SDP) implementation, so that multiple implementations which adhere to the same SD concepts can be compared in experiments. However, the executing SDPs are allowed to generate custom events which will be recorded by ExCovery. Actions that can be executed on participating SD nodes are described as follows.

**Init SD** Mandatory action to allow participation of a node in the SD. Represents “Configuration Discovery and Monitoring”. Discoverable items such and scopes and SCMs are discovered and a unique identity is established on each node. This action reads as parameter the role as either SCM, SU or SM. Optional custom parameters further configure the used SDP. When the SCM parameter is used, the node generates a scm_started event. If an SM registers its service on an SCM node, a scm_registration_add event is generated with the registering node’s identification as parameter. Analogously, when a registration is revoked or changed, the respective events scm_registration_del and scm_registration_upd are generated. In a hybrid architecture, SU and SM agents keep looking for SCMs and emit scm_found events when a SCM has been discovered. When SD initialization is complete, sd_init_done is emitted.

**Exit SD** Stops a previously started role and all assigned searches and publishings, emitting sd_exit_done upon completion. To participate again in the SD process, a node needs to re-run init.

**Start searching** On SU and SM nodes initiates a continuous SD process for a given service type, generating sd_start_search. Refers to the group of “Service-Description Discovery and Monitoring” functions. ExCovery does not distinguish among passive, aggressive, or directed discovery (SCM). A service is considered discovered when its complete description has been received. The event sd_service_add will be emitted with the found service’s identifier as parameter. Analogously, when a service becomes unavailable, the event sd_service_del is generated.

**Stop searching** A previously started search is stopped and event sd_stop_search generated. Includes deregistration of any notification request on SCMs.

**Start publishing** Starts publishing an instance of a given service type, generating a sd_start_publish event. Refers to the group of “Registrations and Extension” functions, such as registration on an SCM.

**Stop publishing** Gracefully stops publishing a given service type. Comprises further actions like aggressively sending revocation messages or SCM deregistration. Generates sd_stop_publish event upon completion.

**Update publication** Updates a previously published service description. Covers underlying functions related to registration on SCMs. Generates an event sd_service_upd with the service identifier as parameter before the update is executed.

An example SD scenario depicted in Figure 9 shows a single active SD in a two-party architecture with a timeline for each actor SU1 and SM1. Actions are shown as white, events as black circles. Unlabeled events inherit the label of the preceding action. In the preparation phase, SU and SM initialize themselves. This phase ends a fixed time after sd_start_publish from SM1 is registered, to let unsolicited announcements of SM1 pass. SU1 then starts...
can be instantiated by programs to analyze, visualize, trace or export experiment related data. As the first platform, ExCover supports the wireless DES testbed at Freie Universität Berlin (FUB) [19]. An implementation for the SD process in Section V exists. This section gives a quick overview of the prototype, a comprehensive description of the implementation can be found in [14].

In accordance to the developed concept the prototype is composed of one controlling entity (master) and a set of controlled entities (nodes) as depicted in Figures 3 and 12. Master and nodes are connected in a centralized client-server architecture with a dedicated communication channel. They communicate synchronously using extensible markup language remote procedure calls (XML-RPC) [20].

The controlling ExperiMaster maintains a list of objects corresponding to the active nodes in the experiment on which actions will be executed. A node object presents the functions of one node to the master program via XMLRPC and uses locking to allow only one access at a time. Which action is executed at which time is specified in process descriptions loaded from the experiment description file. The master creates an EP thread and a fault thread for each abstract node in the description. A single thread is created for the environment manipulations. The actions performed by this thread and the management actions performed by the main program can be executed concurrently on all nodes.

The NodeManager is the central component of the nodes participating in experiments. It handles remote procedure calls (RPCs) from ExperiMaster. Basic procedures exposed via RPC are actions for management, fault injection, environment manipulation and the EP actions as defined in Sections IV and V. The implementation of these functions can be delegated to sub-components, e.g., the EP actions in the context of this work refer to SD actions that are implemented by the avahi software package. Components on a node use the event generator to signal the occurrence of events, as defined in Section IV-B1.

To allow analysis of properties outside the scope of the ExCover processes, for example packet loss and delay, a network packet tagger is provided. It remains running in the background on each node. The tagger adds an option to the header of each selected IP packet and writes a 16 bit identifier to it, incrementing the identifier with each packet.
Additionally, ExCovery includes a set of Python scripts to collect, condition and store experiment results in a database. The presented concept and implementation generally supports multiple SDPs. They need to provide a Linux implementation which provides an interface to fundamental SDP operations, as represented by the actions in Section VII. For the prototype, the Zeroconf SDP suite Avahi [21] was used and modified to allow the association of request and response pairs. This allows analysis of response times not only on SD operation level (\(t_R\) in Figure 9) but on the level of individual SD request and response packets, which by default is not the supported in Zeroconf SDPs. A set of functions exist for extraction and analysis of event and packet based metrics.

VII. CONCLUSION AND OUTLOOK

This work presents the experimentation framework ExCovery to support experiments on the dependability of distributed processes. It provides concepts that cover the description, execution, measurement and storage of experiments, to foster their transparency and repeatability. The description covers the specification of the individual processes of an experiment and their actors: Fault injection, environment manipulation and the main process under experimentation are expressed as interdependent series of actions and events. Execution takes care of controlling the individual nodes during experiment runtime, to make sure each run of an experiment has a clean and defined environment and each node acts according to the experiment description. ExCovery manages series of experiments and recovers from failures by resuming aborted runs. Measurements are taken both on the level of process actions and events and on the level of network packets. Measurements are stored in a unified database format to facilitate sharing and comparison of results.

As a case study, we provide an abstract description of service discovery (SD) as experiment process (EP). ExCovery has been tried and refined in a manifold of SD dependability experiments over the last two years. A working prototype runs on the wireless DES testbed at Freie Universität Berlin.

REFERENCES


