

Taming Orchestration Design Complexity through the ADORE Framework

Sébastien Mosser Mireille Blay-Fornarino

University of Nice – Sophia Antipolis
CNRS, I3S Laboratory, MODALIS Team
{mosser,blay}@polytech.unice.fr

Submission to AOSD'10
forum demonstration track

Abstract

The Service Oriented Architecture (SOA) paradigm supports the assembly of atomic services to create applications that implement complex business processes. Assembly is accomplished by service orchestrations defined by SOA architects. The ADORE framework allows SOA architects to model complex orchestrations of services by composing models of smaller orchestrations involving subsets of services. The smaller orchestrations are called orchestration *fragments* and encapsulates new concerns. ADORE is then used to weave fragments into existing application models. This demonstration illustrates how the ADORE framework can be used to model a SOA, using the Car Crash Crisis Management System (common case study used in a special issue of the TAOSD journal) as leading example.

Problems Addressed

An application in the Service Oriented Architecture (SOA, [6]) paradigm is an assembly of services that implements a business process. SOA applications can be defined as orchestrations of services [12]. A SOA application is typically defined by business specialists and can involve many services that are orchestrated in a variety of ways. Furthermore, the need to extend an SOA application with new business features (to follow market trends) arises often in practice. Existing tools and formalisms (*e.g.* BPMN notation [13], BPEL industrial language [11]) are technologically-driven. They use a *design-in-the-large* approach and considerable effort can be expended when using them to develop and adapt large applications involving many services that are orchestrated in a variety of ways.

We propose a *design-in-the-small* driven framework called ADORE¹ to tame the complexity of orchestration design. Experts focus on the design of small process *fragments*, and let the complexity of composing all the fragments into a final application to dedicated algorithms.

¹Activity moDel supOrting oRchestration Evolution, <http://www.adore-design.org>

Relevance to AOSD

The ADORE framework supports a compositional approach to support complex orchestrations modeling. Models describing smaller orchestrations of services (defined as a set of partially ordered activities) are composed to produce a model describing an orchestration of a larger set of services. The models of smaller orchestrations, called orchestration *fragments*, describe different aspects of a complex business process. ADORE thus allows a business expert to model these concerns separately and then compose them. The automatic support for separation of concerns helps to tame the complexity of developing large business processes in which many concerns must be addressed.

In [3], authors propose a common case study (a Crisis Management System) to compare existing *Aspect Oriented Modeling* approaches between each other. We use it as a leading example in this demonstration, based on our experience acquired while answering this case study [7]. Other experiments made with ADORE (*e.g.*, inductive arithmetic, Web 2.0 folksonomy exploration) are available on the project website.

Uniqueness of design and implementation

To illustrate the ADORE “*uniqueness*”, we propose in this section a very short state-of-the-art overview. Several approaches fill the gap between orchestrations and AOP (*e.g.*, [1], [2]). These approaches rely on the BPEL language and impose to use dedicated BPEL execution engines to interpret the aspects. ADORE preaches technological independence and exposes itself as a *model* to support composition [9]. Instead of interpreting *aspectized* BPEL code, ADORE aims to generate complete orchestrations of services, executable in any industrial engine.

One of the strength of ADORE is to focus on the so-called *Shared Join Points* (SJP, [10]) interactions spawn. We develop a set of rules to identify conflicting interactions between orchestration fragment at composition time. Instead of *re-ordering* the aspects to deal with conflicts around a SJP, we use an *order-independent* composition process. When interactions are detected, the user will enter knowledge at a fine-grained level (where coarse-grained is fragment re-ordering) to solve the conflict and then ease the interaction.

In [4], authors propose a way to weave multiple aspects in UML sequence diagrams. They propose a very precise way to identify join points and express pointcuts, but their weaving methodology relies on a sequential aspect composition, where ADORE uses an order independent composition process.

Inspired by grid-computing community, ADORE proposes an algorithm (fully described in [8]) to automatically enhance a process with *set* concerns. Considering a process p handling a scalar data d , the algorithm can automatically transform p into a process handling a set of data $d^* \equiv \{d_1, \dots, d_n\}$.

Moreover, ADORE allows users or programs to extract information from its internal representation. In this demonstration, we focus on process metrics extraction, inspired by well-known indicators like [5].

Underlying implementation

ADORE user interface is implemented as an EMACS major mode, as shown in Fig. 1. This mode hides in a user-friendly way the set of shell scripts used to interact with the underlying engine.

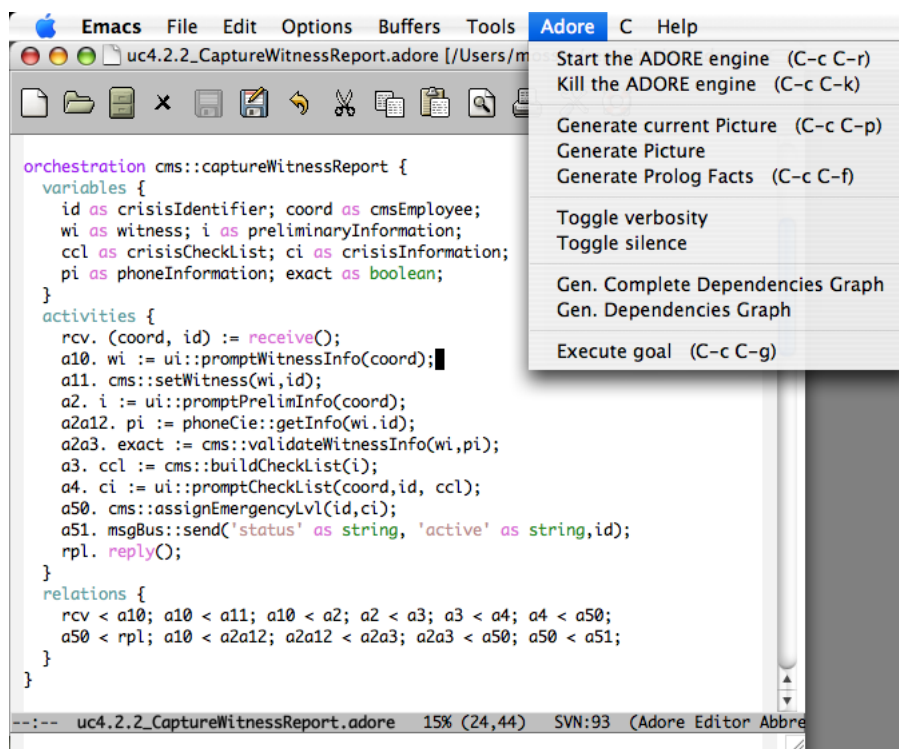


Figure 1: ADORE editor, as an EMACS major mode

The concrete ADORE engine is implemented as a set of logical rules, using the PROLOG language. A dedicated compiler (defined using ANTLR²) implements an automatic transformation between ADORE concrete syntax and the associated PROLOG facts used internally by the engine (Fig. 2). To let the user interact with the engine, we provide a specialized PROLOG *read-eval-print* loop, as shown in Fig. 6.

As visualizing processes is important in design phase, ADORE provides a transformation from its internal facts model to a GRAPHVIZ³ code. This code can then be compiled using the DOT command line tool. It produces as a result a graphical representation of ADORE models, as depicted in Fig. 3. A more concise representation (used to see the impact of composition on the initial process) can be obtained following the same technique (see Fig. 8).

Raw data (*e.g.*, number of activities, relations, process width) can be extracted as a XML document. This document can then be processed (manually

²<http://www.antlr.org/>

³<http://www.graphviz.org/>

```

ADORE facts
:-
createProcess(cms_captureWitnessReport),
setService(cms_captureWitnessReport,cms),
setOperation(cms_captureWitnessReport,captureWitnessReport),
createVariable(cms_captureWitnessReport_id),
setVariableType(cms_captureWitnessReport_id,crisisIdentifier),

```

Figure 2: PROLOG facts, generated by the ADORE compiler

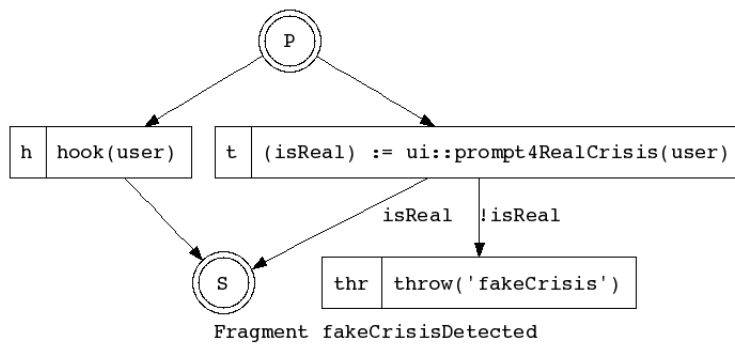


Figure 3: Graphical representation of a fragment in ADORE

or using technology like XSLT) to produce *before/after* graphics and benchmark the approach, as shown in Fig. 4.

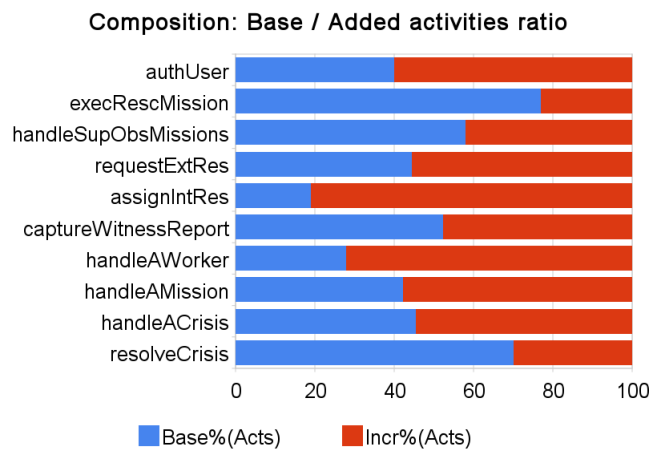


Figure 4: Composition benefits: Before / After activity provenance ratio

Relation to other industrial or research efforts

ADORE was partially funded (2005 – 2009) by the French Research Agency (ANR) through the FAROS consortium. The work of the FAROS consortium (including both industrial– Orange Labs, EDF & Alicante – and academic – IRISA, I3S & LIFL– partners) was to propose a model–driven methodology to build reliable SOA. The ADORE framework is one of the platforms targeted by the FAROS methodology.

Demonstration description

We propose a 4 step demonstration of the ADORE framework, based on the Car Crash Crisis Management System described in [3]. The complete implementation of this example can be found on the project website⁴. Another demonstration was screencasted and published as a FAROS deliverable⁵.

Step #0: Forum presentation. As other *forum* demonstrations, the ADORE demonstration will start by a ten minutes “*slideware*” presentation (including both theoretical foundations and leading example description).

Step #1: Requirements analysis. At this step, we will extract from the CCCMS requirement document associated orchestrations and fragments. We will focus on highlighting reusable requirements and shared join points. Both textual (Fig. 1) and graphical (Fig. 7) representation of ADORE models will be used to illustrate this step.

Step #2: Composition & Engine usage. We will now express a composition unit (fig. 5) and ask the engine to perform the composition. We will then interact with the engine to display the resulting orchestration (both graphical and textual representation) and finally export metrics indicators.

Step #3: Interaction Detection. At this step, we will focus on interaction detection and identify a conflict between two fragments applied around a SJP.

Step #4: Metrics Overview. Finally, we will export from the engine another metrics indicator sets and look at the benefits of the approach.

Demonstration Extra–Information

Contact: The contact person for this demonstration is Sébastien Mosser (mosser@polytech.unice.fr, +334 92 96 50 66).

Hardware requirements: a video–projector, if available.

⁴<http://www.adore-design.org/doku/examples/cccms/start>

⁵http://www.dailymotion.com/video/xal4hu_adore-demonstration_tech

```

fragment fakeCrisisDetected {
  variables { user as cmsEmployee; isReal as boolean; }
  activities {
    h. hook(user);
    t. isReal := ui::prompt4RealCrisis(user);
    thr. throw('fakeCrisis' as string);
  }
  relations { ^ < h; h < $; ^<t; t < $ when isReal; t < thr when ! isReal; }
}

composition cms::captureWitnessReport{
  apply callDisconnected => a10;
  apply callDisconnected => a2;
  apply requestVideo(user: 'coord') => {a3,a4};
  apply ignoreDisconnection => a4;
  apply fakeWitnessInfo => a2a3;
  apply fakeCrisisDetected => a4;
  apply fakeCrisisDetected => requestVideo::a3;
}

```

Figure 5: Fragment & Composition directives in the editor

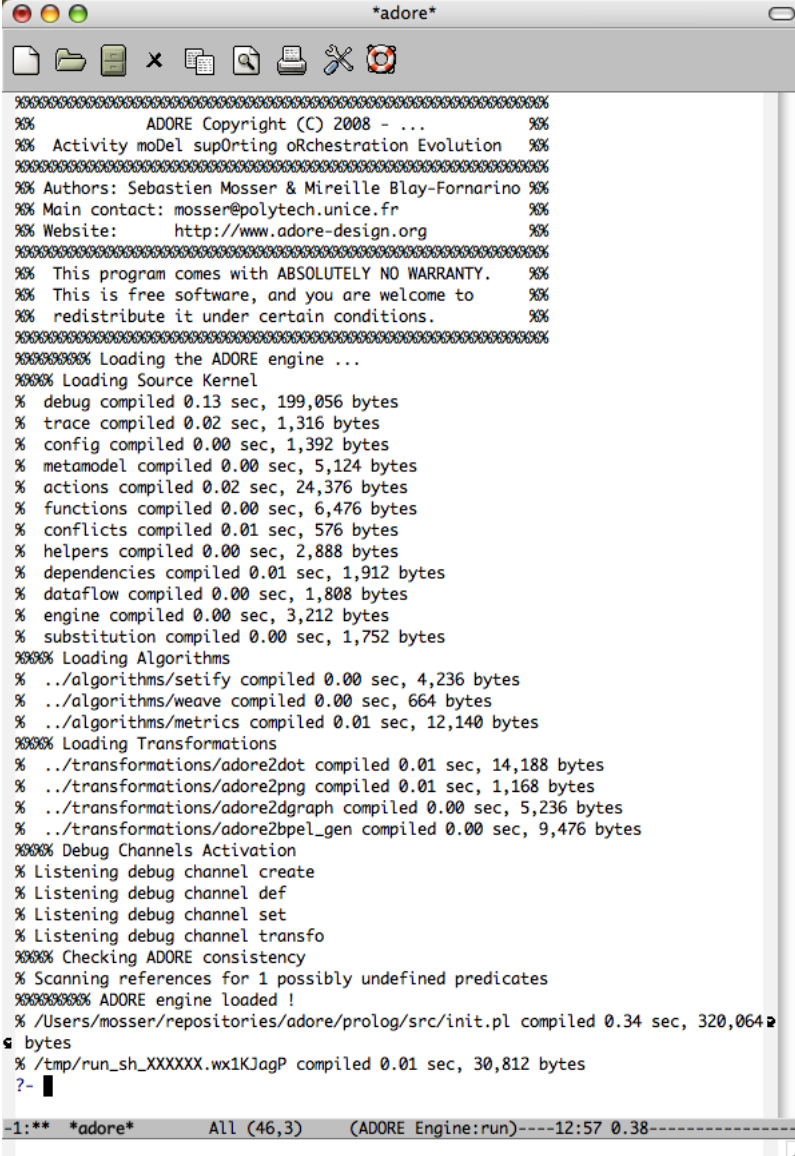
References

- [1] Anis Charfi and Mira Mezini. Aspect-Oriented Web Service Composition with AO4BPEL. In *ECOWS*, volume 3250 of *LNCS*, pages 168–182. Springer, 2004.
- [2] Carine Courbis and Anthony Finkelstein. Weaving Aspects into Web Service Orchestrations. In *ICWS*, pages 219–226. IEEE Computer Society, 2005.
- [3] Jörg Kienzle, Nicolas Guelfi, and Sadaf Mustafiz. Crisis Management Systems, A Case Study for Aspect-Oriented Modeling. Requirements document for taosd special issue, McGill University & University of Luxembourg, September 2009.
- [4] Jacques Klein, Franck Fleurey, and Jean Marc Jézéquel. Weaving Multiple Aspects in Sequence Diagrams. *Transactions on Aspect-Oriented Software Development (TAOSD)*, LNCS 4620:167–199, 2007.
- [5] Ralf Laue and Volker Gruhn. Complexity Metrics for Business Process Models. In Witold Abramowicz and Heinrich C. Mayr, editors, *BIS*, volume 85 of *LNI*, pages 1–12. GI, 2006.
- [6] Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter Brown, and Rebekah Metz. Reference Model for Service Oriented Architecture 1.0. Technical Report wd-soa-rm-cd1, OASIS, February 2006.
- [7] Sébastien Mosser, Mireille Blay-Fornarino, and Robert France. Workflow Design using Fragment Composition (Crisis Management System Design

through ADORE). *Transactions on Aspect-Oriented Software Development (TAOSD)*, pages 1–34, 2010. submitted.

- [8] Sébastien Mosser, Mireille Blay-Fornarino, and Johan Montagnat. Orchestration Evolution Following Dataflow Concepts: Introducing Unanticipated Loops Inside a Legacy Workflow. In *International Conference on Internet and Web Applications and Services(ICIW)*. IEEE Computer Society, May 2009.
- [9] Sébastien Mosser, Mireille Blay-Fornarino, and Michel Riveill. Web Services Orchestration Evolution : A Merge Process For Behavioral Evolution. In *2nd European Conference on Software Architecture(ECSA'08)*. Springer LNCS, September 2008.
- [10] István Nagy, Lodewijk Bergmans, and Mehmet Aksit. Composing Aspects at Shared Join Points. In Robert Hirschfeld, Ryszard Kowalczyk, Andreas Polze, and Mathias Weske, editors, *NODE/GSEM*, volume 69 of *LNI*, pages 19–38. GI, 2005.
- [11] OASIS. Web Services Business Process Execution Language Version 2.0. Technical report, OASIS, 2007.
- [12] Chris Peltz. Web Services Orchestration and Choreography. *Computer*, 36(10):46–52, 2003.
- [13] Stephen A. White. *Business Process Modeling Notation (BPMN)*. IBM Corp, May 2006.

Appendixes: Screenshots



```
*****
%%      ADORE Copyright (C) 2008 - ...      %%
%% Activity model supporting orchestration Evolution %%
*****
%% Authors: Sebastien Mosser & Mireille Blay-Fornarino %%
%% Main contact: mosser@polytech.unice.fr %%
%% Website: http://www.adore-design.org %%
*****
%% This program comes with ABSOLUTELY NO WARRANTY. %%
%% This is free software, and you are welcome to %%
%% redistribute it under certain conditions. %%
*****
%% Loading the ADORE engine ...
%% Loading Source Kernel
% debug compiled 0.13 sec, 199,056 bytes
% trace compiled 0.02 sec, 1,316 bytes
% config compiled 0.00 sec, 1,392 bytes
% metamodel compiled 0.00 sec, 5,124 bytes
% actions compiled 0.02 sec, 24,376 bytes
% functions compiled 0.00 sec, 6,476 bytes
% conflicts compiled 0.01 sec, 576 bytes
% helpers compiled 0.00 sec, 2,888 bytes
% dependencies compiled 0.01 sec, 1,912 bytes
% dataflow compiled 0.00 sec, 1,808 bytes
% engine compiled 0.00 sec, 3,212 bytes
% substitution compiled 0.00 sec, 1,752 bytes
%% Loading Algorithms
% ../algorithms/setify compiled 0.00 sec, 4,236 bytes
% ../algorithms/weave compiled 0.00 sec, 664 bytes
% ../algorithms/metrics compiled 0.01 sec, 12,140 bytes
%% Loading Transformations
% ../transformations/adore2dot compiled 0.01 sec, 14,188 bytes
% ../transformations/adore2png compiled 0.01 sec, 1,168 bytes
% ../transformations/adore2dgraph compiled 0.00 sec, 5,236 bytes
% ../transformations/adore2bpel_gen compiled 0.00 sec, 9,476 bytes
%% Debug Channels Activation
% Listening debug channel create
% Listening debug channel def
% Listening debug channel set
% Listening debug channel transfo
%% Checking ADORE consistency
% Scanning references for 1 possibly undefined predicates
%% ADORE engine loaded !
% /Users/mosser/repositories/adore/prolog/src/init.pl compiled 0.34 sec, 320,064
s bytes
% /tmp/run_sh_XXXXXX.wx1KJagP compiled 0.01 sec, 30,812 bytes
?-
-1:** *adore* All (46,3) (ADORE Engine:run)---12:57 0.38-----
```

Figure 6: ADORE engine, as a *read-eval-print* query loop

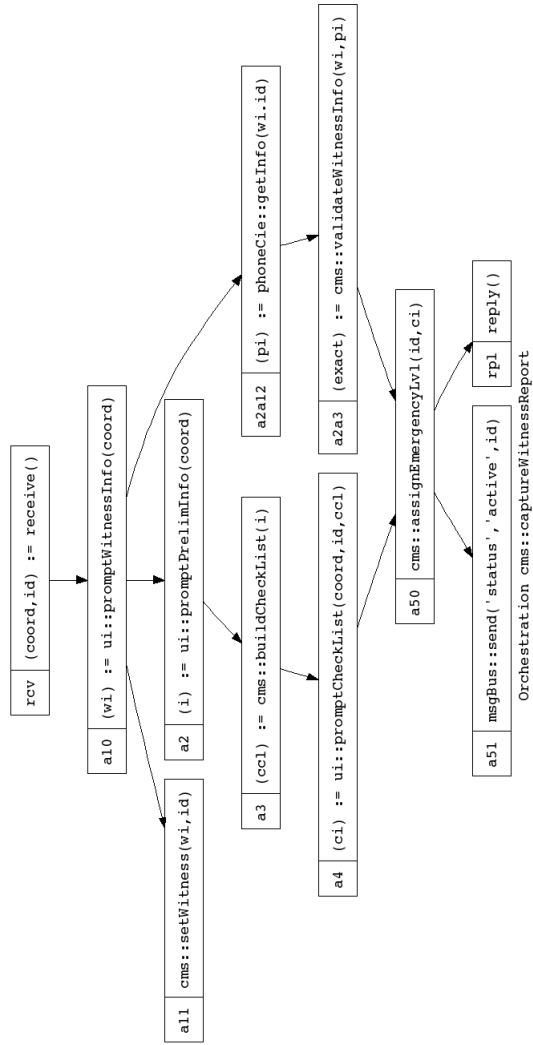


Figure 7: Graphical representation of an orchestration in ADORE

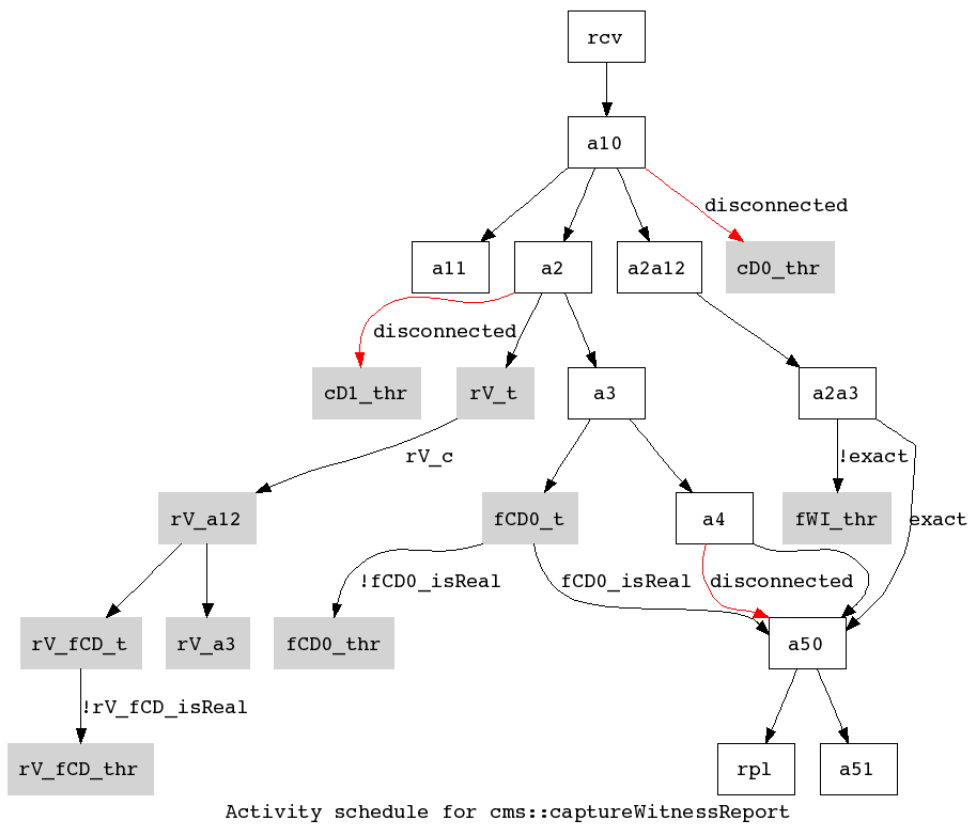


Figure 8: Composition dashboard (grey activity were added by composition)