Robust Design of a Collaborative Platform for Model-Based System Engineering: Experience from an Industrial Deployment

Christophe Ponsard[1(✉)], Robert Darimont[2], and Mounir Touzani[3]

[1] CETIC Research Center, Charleroi, Belgium
christophe.ponsard@cetic.be
[2] Respect-IT SA, Louvain-la-Neuve, Belgium
robert.darimont@respect-it.be
[3] Toulouse, France

Abstract. Model-Based System Engineering is gaining momentum in the industry. In order to be successful, it requires adequate tooling support. In addition to functional requirements related to model edition, verification and transformation, key non-functional requirements need to be carefully addressed such as versioning, usability/team work, reliability, security, ease of integration. In this paper, we first give an overview of how we dealt with such requirements in the context of the development of a real world platform for a global telecom operator, with a focus on early steps of system modelling. We then present a more detailed design of the tooling architecture and a high availability protocol for accessing a mainstream model repository. The proposed protocol is modelled and verified using the Alloy language and model-checker.

Keywords: Model-Based System Engineering · Tool support · Modelling · Industrial transfer · High availability · Alloy · Model-checking

1 Introduction

Modelling has been used for a long time across many engineering disciplines like civil engineering, electronic systems and aeronautics. It is now increasingly applied at system level across disciplines through Model-Based System Engineering (MBSE) with the aim to rely primarily on domain models to support the exchange between engineers rather than documents. Model-Driven Engineering (MDE) is a similar trend focusing only the software development process [30]. Such approaches can rely on standardised and well adopted modelling languages like SysML [22] at system level, UML [21] for software and increasingly Domain

M. Touzani—Independent Researcher.

K.-D. Schewe and N. K. Singh (Eds.): MEDI 2019, LNCS 11815, pp. 333–347, 2019.
https://doi.org/10.1007/978-3-030-32065-2_23

Specific Languages (DSLs). They provide a visual syntax enabling the design and communication activities of the engineers but also have precise semantics to enable automation of parts of the System Development Life Cycle (SDLC).

Efficient modelling across the different engineering activities can only be achieved based on reliable computer tools typically composed of a modelling environment, a model repository and a model transformation toolchain for synchronising modelling artefacts at different steps of the SDLC. Model-to-model and model-to-text transformations are respectively used to generate detailed models from abstract ones and code/documentation from models.

Designing a robust toolchain at industrial level is not an easy task. In addition to functional requirements (FR) relating to various model manipulations (edition, check, transformation, simulation,...), it is also very important to cover several non-functional requirements (NFR), in order to ensure industrial strength. Frequently cited NFR are usability, support for collaboration and versioning, scalability, highly availability, integrity, confidentiality, interoperability and maintainability [27,29,31].

In this paper, we present an industrial feedback to cope with such non-functional requirements by elaborating a MBSE platform for a global telecom operator (Huawei Ltd). Our tooling is focusing on the early steps of system development through a goal-oriented approach relying on elaborated requirements modelling. The contribution of this paper is twofold:

- First, we give a high level view about how we addressed important NFR without focusing too much on the specifics of our industrial case but rather by trying to provide adequate feedback that can be applied in a wider context.
- Second, we focus on robust operation requirements, i.e. high availability and load balancing, by describing a generic architecture composed of several redundant server nodes able to process multiple requests and reconfigure in case of node failure. This protocol also involves a master node with specific responsibilities, which must be reassigned to another node in case of failure.

The reported case was carried out over two years, with the last six months mainly devoted to enforcing the robustness of the platform. It is more extensively detailed from a requirements engineering perspective in [25].

This paper is structured as follows: Sect. 2 presents our industrial case and analyses its key requirements. Then Sect. 3 elaborates on how we dealt with non-functional requirements with a generalisation effort. Section 4 goes into details about the specific high availability requirement. It presents a multi-server architecture and a specific protocol ensuring robust operation in presence of node failures. It is modelled and verified using the Alloy language and analyser. Section 5 discusses some related work. Finally, Sect. 6 draws some conclusions and presents our future work.

2 Presentation of Our Industrial Case

This section gives a summary of the context and main requirements of the developed platform. We try to step away from too specific aspects of the industrial

case that initiate the work in order to provide a more general feedback. Another reason is that the core of the resulting platform already proved applicable in other domains. An extended description of the Huawei deployment is available [25] and a demonstration version accessible at: http://demo.objectiver.cetic.be/objectiver/client.

2.1 Context and Objectives

The context of our case is quite common to many industries with requirements engineering practices mainly based on domain modelling, analysis of the current solution (i.e. existing product), use case analysis and UML/SysML modelling. The global process is still strongly document-based with different types of documents flowing across the lifecycle. Domain specific languages were already present, mainly for design and testing phases. For example, Gherkins was used to formalise specifications using "Given-When-Then" structure which can be used for testing [3,36].

A common long term objective of companies is also to evolve towards a wider use of modelling across the SDLC but also across products, i.e. by modelling their product lines and using it for better reuse through more systematic domain engineering. However, this evolution should be progressive and preserve the current flow of documents. The transition can be achieved the efficient production of documents using model-to-text [23]. Later on, some documents could become obsolete when direct model-to-model integration is achieved [13].

2.2 Key Requirements

As we focus on requirements modelling, the starting point was to obtain an adequate meta-model for capturing all the knowledge related to stakeholders goals, system properties, domain properties, assumptions on users, and information to be exchanged. As the meta-models available in standard modelling languages such as UML and SysML are mostly poor with this respect, a specialised meta-model was selected: KAOS [7,14], among other candidates like i* [37] or URN [5]. In addition to concepts, different sources and targets of the model transformations were also modelled, like diagrams or documents. Possible transformations between those artefacts were also identified and are documented in [26]. For example, requirements can be tagged in a source document and refined using decomposition inside a diagram, then selected as part of a specific subsystem and exported in a public tender or directly transferred in the development department of the company.

In order to minimise the effort to build a model and to maximise the value from the invested modelling effort, the proposed tooling needs to have the following qualities (or NFR):

– *Scalability*: efficient support for large models but also for several models and multiple concurrent users.

- *High availability*: system up and running with very reduced unplanned interrupt time, meaning service reliability and server redundancy.
- *Navigation across multiple versions* of modelling artefacts for traceability or better collaboration support.
- *Usability* (visual feedback, shortcuts,...) for productivity and adoption.
- *Flexible integration*: to exchange models or expose specific (web-)services.
- *Security* enforcement (model integrity, confidentiality, access control).
- *Long term support/portability*: to ease maintenance over a long time and to enable reuse through a knowledge base or product lines.
- *Reduced installation and maintenance* effort to minimise operation costs.

3 Dealing with Non-functional Requirements

3.1 Global Architecture

Several NFR can be addressed through an adequate tool architecture. Our architecture is depicted in Fig. 1. We give here a short summary why it is convenient. More information is available in [8].

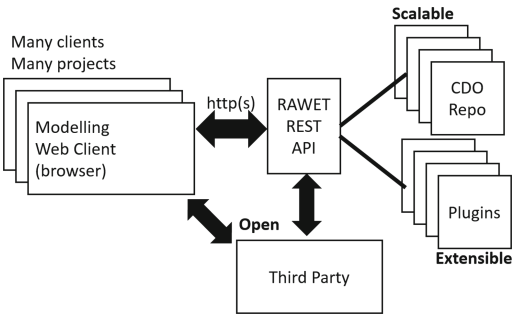

Fig. 1. Global platform architecture

Our tooling architecture is composed of:

- several clients, including a full web-based client, either standalone or embedded in third party tools.
- a RESTful API, called RAWET, providing services for model and diagram edition, history, snapshots, user authentication and project management. It also enables different kinds of integration [28].
- a back-end composed of the model repository relying on a Eclipse Modelling Framework (EMF) store [33] and a collection of plugins enabling both web-services and user interface extensions.

3.2 Scalability and High Availability

Scalability and high availability are crucial for the industrial adoption of an MBSE tooling. This are dealt with at the architecture level an more specifically the model repository which must be able to manage a large number of models, possibly large in size.

In our case the Eclipse Modelling Framework is used [32]. Different solutions to persist EMF are available and the selected one, Connected Data Object (CDO), offers different possible back-ends, including a mature and scalable relational database manager also with mirroring capabilities.

The server itself is dealing with our RAWET service API. Standard web application technologies can be used to dispatch requests on many concurrent servers and, at the same time, allow some server to be down, thus addressing both scalability and high availability of the service. However, our architecture requires that the model repository access is centralised on a single server which is thus a possible point of failure of the system. In order to cope with this problem, we designed a specific protocol, which is detailed in Sect. 4.

3.3 Ease of Integration

Toolchain integration has started from simple import/export mechanisms and evolved towards a more complex integration with specific tools such as text processors and other SDLC tools. A key decision was to shape the tooling as a series of services available for use over the company intranet. This comes at two different levels:

– *at the user interface level*, the tool provides a similar experience as other modelling tools. However, due to its modular design, web-client extensions

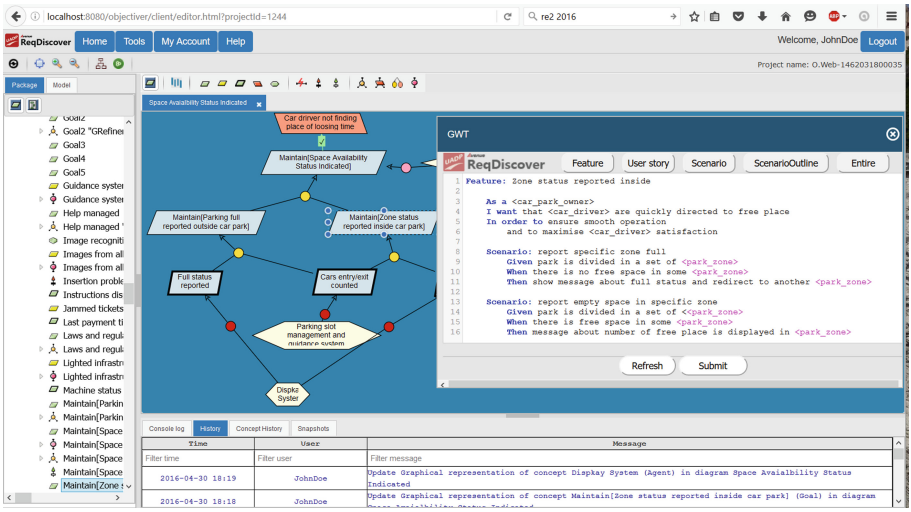

Fig. 2. Modular web-based user interface

can easily be embedded. For example, Fig. 2 shows the integration of an editor for the GWT Domain Specific Language. Conversely, specific components can use reused inside other tools, e.g. a dashboard or read-only view.
– *at the model level*, a clean RESTful API is directly available to perform CRUD (Create/Read/Update/Delete) operations both on the model elements and on model representations inside diagrams, baselines, etc. This allows third-party tools to directly push or query requirements inside the tool while, previously, many import and export actions had to be initiated from the tool.

3.4 Usability

Usability was largely stressed by our Chinese customer. The standard model edition features had to be enriched with extensions in order to:

– provide quick access to frequently used features, with minimal number of clicks and even keyboard shortcuts.
– support batch operation over multiple concepts (e.g. move, change type).
– tune graphical representation of concepts based on meta-model extensions (e.g. through decorations on such extended concepts).
– provide efficient default graphical layout and include filtering capabilities.

3.5 Versioning

Model versioning is required to track the model evolution. Versioning is supported by the CDO model repository [33]. However, the provided technical features had to be translated to a more intuitive user experience. Our implementation started with the support of a single baseline and was extended to multiple baselines with comparison and rollback capabilities. Access to concepts history was also made easily accessible at the user interface level to ease collaboration.

4 Analysis of the High-Availability Protocol

This section studies the robustness of server operation. The server is taking care of model manipulation initiated from the client side and implemented through a well-defined API. Its implementation can be assumed stateless because in case of crash, a server session can easily be restarted without impacting the client.

A standard solution for increasing availability and coping with high load is to use multiple servers and a load balancer/monitor front-end service, like NGINGX [34]. A typical deployment with three servers is depicted in Fig. 3(a). An important constraint relates to the access to the model repository: each server actually maintains a form of cache of modelling concepts related to its user sessions. However, all the traffic to the model repository needs to be processed by a single node which is the gateway to the model repository. This node ensures the serialisation of changes and notifies all other nodes of the relevant changes through a synchronisation mechanism. As this node has a specific role, we call

it *master* in our architecture. Some other unique responsibilities may also be assigned to this node.

As the master is different from the other servers, its failure cannot be resolved by simply redirecting the traffic to another node as this is the case for non-master nodes as described in Fig. 3(b).

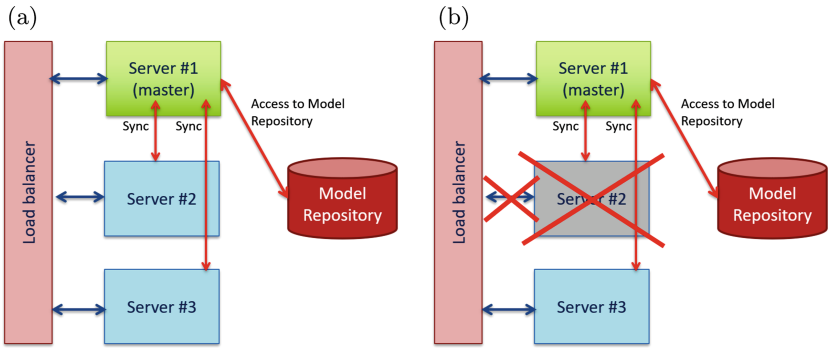

Fig. 3. (a) Fully operational system (b) Failure of a non-master server

4.1 Informal Model for Master Recovery

As depicted in Fig. 4(a), in case of crash of the master server, the access to the repository is also lost and the whole system is going to freeze until a new master is restored. Hence, it must happen quickly.

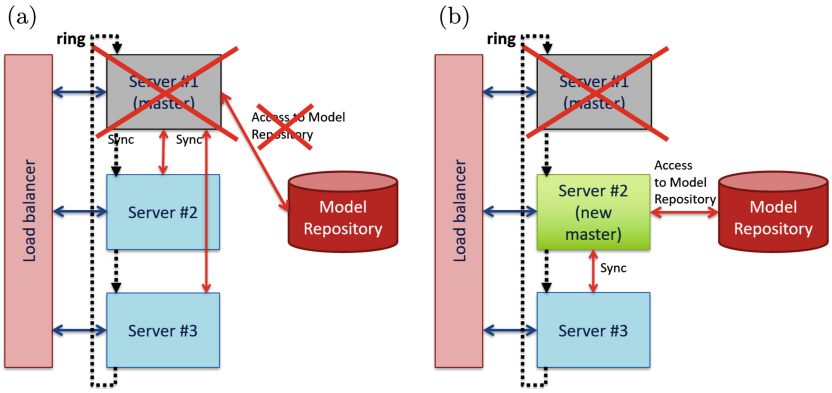

Fig. 4. (a) Failure of the master server (b) Recovery (new master election)

For finding a new master, we will consider the nodes form a logical ring, e.g. from the lower to highest IP address and then back to the lowest one. Given that setting, we could apply a standard leader election protocol such as Chang

and Roberts [6]. However, we do not restrict communication to message passing between adjacent nodes. We also need to consider the sub-ring formed by the working servers because, in our algorithm, we choose to select as new master, the first available server following the crashed master. This master will then reconnect to the database and start synchronising will all remaining servers. This will result in the new operational situation depicted in Fig. 4(b) where server#2 is the new master. Note that is the crashed master is restarted, it will start acting as a normal server and will synchronise with master server#2.

In order to achieve this, the following rules are applied: a server detecting a master crash will start scanning the previous nodes in the logical ring

- until finding a running server, then this server has priority to become master and the server will wait for this new master to come up and contact him.
- or finding the crashed master, then the server is the first alive server after the master and it should become the master. It will then notify all nodes it is now the master. All other nodes will then reinitialise their synchronisation link with the new master.

Given the load-balancer only directs request to working nodes, if the master is crashed, the request must go through another server and this will trigger the change of master as soon as there is a client request. Periodic monitoring requests internal to the platform can also be used to avoid waiting for a client request.

4.2 Formal Modelling with Alloy

In order to make sure our protocol is behaving as expected, we decided to model it and verify it using a formal modelling. We selected Alloy because it is a lightweight formal method [10,12]. On the tool side, the Alloy Analyser relies on model-checking, which is fully automated contrary to theorem proving, and has a nice way for visualising solutions with many filtering and rendering capabilities [11]. This section first describes the static part of the model, then its behaviour and finally, different validation experiments.

4.3 Structure of the System

The system structure is described in Listing 1.1. It relies on *Time* and *Server* signatures which are ordered using the available ordering module. The ring structure is enforced using a circular constraint on the time-independent *succ* attribute. Three other time-dependent attributes are used: *crashed*, which records at which time a server was crashed, *master*, which records at which time a server was a master and *link*, which records who each server believes is the master at a given time. A key requirement that needs to be checked, is that at any given time only one master may exist.

Listing 1.1. Structure of the System

```
open util/ordering[Time] as TO     -- time steps is ordered
open util/ordering[Server] as SO   -- processes are ordered

sig Time {}                        -- Time steps (ordered)

sig Server {                       -- Server node (ordered)
  succ: Server,                    -- this is a static topology
  crashed: set Time,               -- captures when a server is crashed
  master: set Time,                -- captures when a server is master
  link: Time -> lone Server        -- captures knowledge of a node about master
}

fact ring {                        -- Server nodes are constrained to form a ring
  all p: Server | Server in p.^succ
}
```

4.4 Dynamic Modelling for Maintaining Master Node

In order to build a dynamic model, we use standard Alloy modelling guidelines [9], i.e. we define a trace composed of sequence of *Time*, starting with some initialisation *init* with no crashed server and the master allocated on the first one. Each pair of successive *Time* of a trace is constrained to be either a *normalOperation* when the *master* to be up and running, or a *recoverOperation* when this is not the case. The *masterAvailable* predicate is used for this test.

In *normalOperation*, the state is globally unchanged: a crashed node remains crashed (repair is considered later) but we allow new nodes to crash, so we can study server unreliability. However, we do not allow all the nodes to crash (see *notFullyCrashed* predicate) because in that case no solution is possible.

In *recoveryOperation*, the first non-crashed successor of the crashed master is selected as new master. This node is identified in a single *Time* step through a transitive closure on the *succ* function with domain and range filtering to discard crashed node. All other Servers are then informed of this new master by directly changing their *link* relationship. Listing 1.2 presents the full specification of this behavioural part.

Listing 1.2. Behaviour of the System

```
pred init [t: Time] {
  t in SO/first.master
  all s:Server |
    (t ∉ s.crashed)
    and (s≠SO/first =\textgreater t ∉ s.master)
    and (s.link[t]=SO/first)
}

pred masterAvailable(t:Time) {
  all s:Server |
    t in s.master =\textgreater t ∉ s.crashed
}

pred notFullyCrashed(t:Time) {
  some s:Server | t ∉ s.crashed
}
```

```
pred normalOperation [t, t': Time, s: Server] {
  masterAvailable[t]
  t in s.crashed =\textgreater t' in s.crashed  -- crashed stuff remains so
  --  but note that new crash may occur !
  in s.master iff t' in s.master  -- nothing changed about master routing
  s.link[t']=s.link[t]            -- nothing changed about master routing
  notFullyCrashed[t']             -- restricting fault model
}

pred recoverOperation [t,t': Time, s: Server] {
  not masterAvailable[t]
  let select=(^(crashed.t <: succ)) :> (Server-crashed.t) | -- new master !
    (t' in s.master <>  select[s.link[t]]=s)  -- new master
        and s.link[t']=master.t.(select)          -- updating links
  t' in s.crashed iff t in s.crashed           -- no crash during recovery
}

fact traces {          -- fact for constraining traces to allowed
    ↪ operations
  init [first]
  all t: Time-last | let t' = t.next | all s:Server |
     normalOperation [t, t', s] or recoverOperation [t, t', s]
}
```

4.5 Model Validation

Prior to model-checking, it is important to validate the consistency of the model, i.e. that it has instances and that those instances match the intended behaviour. In order to validate our model, we first look for *SingleMasterCrash* in cascade, i.e. each time there is a master, it should be crashed the next time, as this is allowed by our *normalOperation*. The expected behaviour is that the next server should take over as master and then crash, hence the master server will progress around the ring. Note that because crashed nodes remains crashed, no instance will be possible if there are more time steps than the double of the size of the ring. The resulting scenario is depicted in Fig. 5(a) for the three first time steps and it behaves as expected.

Listing 1.3. Behaviour of the System

```
-- find some instance with a lot of server crashing
pred singleMasterCrash { all t:Time | all s:Server |
  t in s.master => t.next in s.crashed }
run singleMasterCrash for 5 Server, 8 Time
-- find some instance with a lot of server crashing and first backup node
    ↪ too
pred MasterAndBackupCrash {  all t:Time | all s:Server |
  t in s.master => (t.next in s.crashed and t.next in s.succ.crashed) }
run MasterAndBackupCrash for 5 Server, 5 Time
```

A second validation is more naughty and involves the simultaneous crash of the master and its backup node (i.e. immediate successor). In this case, we expect the second successor server of the master to take over. The trace in Fig. 5(b) (limited to the three first time steps here) shows this is the observed behaviour.

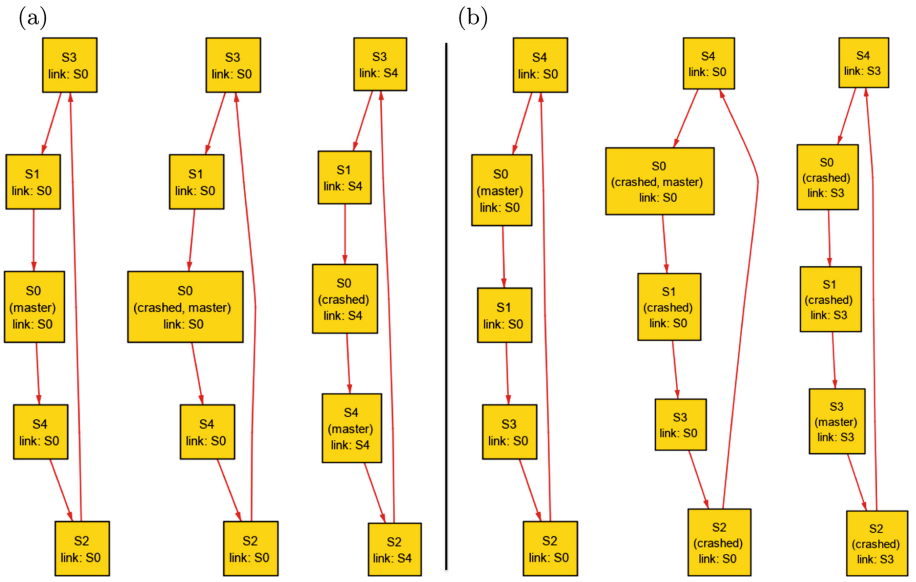

Fig. 5. Behaviour in case of (a) single failure (b) double failure

4.6 Model Checking

Finally, we can ask the analyser to verify the uniqueness of the master at all times. Rather than requiring exactly one master, actually two separate verifications are performed: at least one master and at most one master (see Listing 1.4). Their conjunction yields the wished property, but each kind of violation is more interesting to study separately.

Listing 1.4. Behaviour of the System

```
-- no multiple masters allowed
assert AtMostOneMaster { all t: Time | lone s: Server | t in s.master }
check AtMostOneMaster for 5 Server, 15 Time

-- at least one master allowed
assert AtLeastOneMaster { all t: Time | some s: Server | t in s.master  }
check AtLeastOneMaster for 5 Server, 15 Time
```

Listing 1.5 recapitulates the running time of all the checks performed on a core I7 laptop with a 64 bit Java Virtual Machine. One can see the validation are straightforward, meaning it is easy to find instances of the model, while the verification took much longer: about 5 s for *AtMostOneMaster* and about 25 s for *AtLeastOneMaster* for 5 servers and 15 units of *Time*. The verification did not find any counter-example meaning the model might be valid. Given the limited variety of scenarios, one might be confident the system is indeed correct. However, the behaviour should be studied in further details by removing some of the limitations:

- by allowing crashed servers to become operational again during operation mode. In this case, the verification is still fine but takes more time for *AtLeastOneMaster* (about 2 min)
- by allowing failures during the restoration step. In this case, there can be scenarios without any master beyond a given step after a crash of all servers. When excluding fully crashed states, the verification is fine too.

Listing 1.5. Run result

```
Executing "Run singleMasterCrash for 5 Server, 8 Time"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   7590 vars. 305 primary vars. 21359 clauses. 31ms.
   Instance found. Predicate is consistent. 32ms.
Executing "Run MasterAndBackupCrash for 5 Server, 5 Time"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   4773 vars. 200 primary vars. 12892 clauses. 22ms.
   Instance found. Predicate is consistent. 31ms.
Executing "Check AtMostOneMaster for 5 Server, 15 Time"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   14562 vars. 565 primary vars. 41731 clauses. 62ms.
   No counterexample found. Assertion may be valid. 4919ms.
Executing "Check AtLeastOneMaster for 5 Server, 15 Time"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   14554 vars. 565 primary vars. 41729 clauses. 63ms.
   No counterexample found. Assertion may be valid. 25207ms.

4 commands were executed. The results are:
   #1: .singleMasterCrash is consistent.
   #2: .MasterAndBackupCrash is consistent.
   #3: No counterexample found. AtMostOneMaster may be valid.
   #4: No counterexample found. AtLeastOneMaster may be valid.
```

5 Related Work and Discussion

Several web-based tools are available to develop diagrams in different notations such as UML, BPMN and flowcharts [4,15,17,18]. They provide an easy way to draw diagrams from a web browser without requiring any installation, to save them in the Cloud and to share access with other team members. Although some may be based on Open Source [4], they all adopt a Software as a Service (SaaS) model with pay-per-use beyond a limited basic offer, e.g. to support larger models, more concurrent users, or tool integration. The majority of those tools focus on the graphical notations and do not stress the model behind them, nor the API to be able to access that model. However, some tools provide such an API, for example GenMyModel has an API to return user information, project details, execute project commands, return project tag data, and more [16]. Cacoo provides a quite similar API [19]. However when testing those tools and analysing their API, it appears that many of them have a weak notion of model, i.e. concept and their representation are not distinguished making impossible to share concept across diagrams. This also limits the ability to feed the model into a MDE toolchain. An exception is GenMyModel which also provides EMF import and export. Our approach is close the GenMyModel as we support a strong notion of model and provide an RESTFul API with all the usual CRUD operations on

model concepts and representations. Beyond this, we also support project/user level operations and more advanced operations, for example to manage model versioning.

Our approach relies on the EMF Open Source modelling frameworks which is actually widespread in the research community but less in the industrial world where the majority of modelling tools are Closed Source, e.g. Rhapsody, MagicDraw and Enterprise Architect. This means such tools are missing recent advanced made by research tools. Our work aims at bridging this gap by enabling different forms of integration. Other researchers have also explored how to address this problem through mechanisms going beyond the pure exchange of models in standard formats like XMI [24] or through protocols like OSLC [20]. An attempt to bridge a proprietary UML modelling tool (PTC Integrity Modeller) with an Open Source family of languages for automated model management (Epsilon) is discussed in [38]. The question is also crucial in Cyber Physical Systems to support model integration across domains. OpenMETA was applied for the design and implementation of an experimental design automation tool suite [35]. It could provide multiple level of abstraction, correctness-by-construction in an heterogeneous context and reuse of Open Source tools assets. Our work is following the same design principles but with a bigger priority on tool reliability and availability.

6 Conclusion and Future Work

In this paper, we first investigated key non-functional requirements for building a MBSE toolchain based on our industrial experience, focusing on the early analysis steps. Although our work is driven by a specific case, the identified NFR are of general nature and are also reported by others in the literature. So we believe our feedback can be useful in other cases. Then, we focused on the specific NFR of high-availability in the context of pool of servers with a single repository access. We proposed a design to maintain a master node in a reliable way by modelling and verifying our design using the Alloy analyser. Although our solution was developed in the context of an EMF data store, we believe that the problem is more general in nature and that our solution can be reused.

In our future work, we plan to keep improving availability by also investigating problems on the repository and the load-balancing components, e.g. through mirroring or mutual monitoring. We also plan to refine our model at a finer level of operation (i.e. message level). For example, our model does not capture the behaviour when a server is crashing during the notification phase. Our intent is also to investigate another formal method supporting model refinement, such as Event-B and the Rodin toolkit [1,2]. We also plan to further analyse security requirements, which were not within the scope of our initial work because the tool was deployed within a secured intranet.

Acknowledgements. This research was partly supported by the SAMOBIGrow project (nr. 1910032). We thank Respect-IT and Huawei for their feedback in the elaboration of this tooling. We also thank the reviewers for their comments.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, New York (2010)
2. Abrial, J.R., et al.: Rodin: an open toolset for modelling and reasoning in event-B. STTT **12**(6), 447–466 (2010)
3. Adzic, G.: Specification by Example: How Successful Teams Deliver the Right Software, 1st edn. Manning Publications Co., Greenwich (2011)
4. Alder, G., Benson, D.: draw.io (2011). https://about.draw.io/integrations
5. Amyot, D., Mussbacher, G.: User requirements notation: the first ten years, the next ten years. JSW **6**(5), 747–768 (2011)
6. Chang, E., Roberts, R.: An improved algorithm for decentralized extrema-finding in circular configurations of processes. Commun. ACM **22**(5), 281–283 (1979)
7. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Program. **20**(1–2), 3–50 (1993)
8. Darimont, R., Zhao, W., Ponsard, C., Michot, A.: A modular requirements engineering framework for web-based toolchain integration. In: 24th IEEE International Requirements Engineering Conference, RE 2016, Beijing, China, 12–16 September, pp. 405–406 (2016)
9. Dennis, G., Seater, R.: Alloy Analyzer 4 Tutorial Session 4: Dynamic Modeling Software. Design Group. MIT (2017)
10. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. **11**(2), 256–290 (2002)
11. Jackson, D.: Alloy Analyser, Version 4 (2006). http://alloytools.org
12. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press, Cambridge (2012)
13. Kahani, N., Bagherzadeh, M., Cordy, J.R., Dingel, J., Varró, D.: Survey and classification of model transformation tools. Softw. Syst. Model. **18**(4), 2361–2397 (2018)
14. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, Hoboken (2009)
15. Legrand, T.: Genmymodel (2012). https://www.genmymodel.com
16. Legrand, T.: GenMyModel API Documentation (2014). https://api.genmymodel.com/doc
17. Lucid Software: Lucidchart (2008). https://www.lucidchart.com
18. Nulab Inc.: Cacoo (2009). https://cacoo.com
19. Nulab Inc.: Cacoo API Overview (2012). https://developer.nulab.com/docs/cacoo
20. OASIS: Open Services for Lifecycle Collaboration (2008). https://open-services.net
21. OMG: Unified modeling language (1997). http://www.omg.org/spec/UML
22. OMG: System modeling language (2005). http://www.omg.org/spec/SysML
23. OMG: MOF Model to Text Transformation Language (2008). http://www.omg.org/spec/MOFM2T
24. OMG: XML Metadata Interchange v2.5.1 (2015). https://www.omg.org/spec/XMI
25. Ponsard, C., Darimont, R.: Improving requirements engineering through goal-oriented models and tools: feedback from a large industrial deployment. In: Proceedings of 12th International Conference on Software Technologies, ICSOFT, Madrid, Spain, 24–26 July 2017
26. Ponsard, C., Darimont, R., Michot, A.: Combining models, diagrams and tables for efficient requirements engineering: lessons learned from the industry. In: INFORSID 2015, Biarritz, France, June 2015

27. Ponsard, C., Deprez, J.C., Delandtsheer, R.: Is my formal method tool ready for the industry? In: 11th International Workshop on Automated Verification of Critical Systems, Newcastle, UK, 12–14 September 2011
28. Ponsard, C., Michot, A., Darimont, R., Zhao, W.: A generic rest API on top of eclipse CDO for web-based modelling. EclipseCon France, Toulouse, June 2016
29. Ryan, M., Cook, S., Scott, W.: Application of MBSE to requirements engineering research challenges. In: Systems Engineering, Test and Evaluation Conference SETE2013, Canberra, Australia, April 2013
30. Schmidt, D.C.: Guest editor's introduction: model-driven engineering. Computer **39**(2), 25–31 (2006). https://doi.org/10.1109/MC.2006.58
31. Soukaina, M., Abdessamad, B., Abdelaziz, M.: Model driven engineering tools: a survey. Am. J. Sci. Eng. Technol. **3**(2), 29 (2018)
32. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0, 2nd edn. Addison-Wesley Professional, Upper Saddle River (2009)
33. Stepper, E.: Connected data object (2006). https://www.eclipse.org/cdo
34. Sysoev, I.: Nginx (2004). https://nginx.org
35. Sztipanovits, J., et al.: Model and tool integration platforms for cyberphysical system design. Proc. IEEE **106**(9), 1501–1526 (2018)
36. Wynne, M., Hellesoy, A.: The Cucumber Book. The Pragmatic Programmers. Pragmatic Bookshelf, Dallas (2012)
37. Yu, E.S.K., Mylopoulos, J.: Enterprise modelling for business redesign: the i* framework. SIGGROUP Bull. **18**(1), 59–63 (1997)
38. Zolotas, A., et al.: Bridging proprietary modelling and open-source model management tools: the case of PTC integrity modeller and epsilon. In: Software & Systems Modeling (2019)