Inf Syst Front (2016) 18:891–908
DOI 10.1007/s10796-016-9663-9

Semantic weaving of configuration fragments into a consistent system configuration

Azadeh Jahanbanifar[1] · Ferhat Khendek[1] · Maria Toeroe[2]

Published online: 13 June 2016

Abstract Configuration fragments developed separately and focusing on different aspects, such as availability, security or performance of a system need to be integrated into a consistent system configuration to avoid system malfunctions. The main challenges of such integration are due to the overlapping entities and the integration relations between the entities of the different configuration fragments. In this paper we propose a model based approach for a consistent integration of configuration fragments into a system configuration. We use and extend the model weaving technique to capture the semantics of the relations between the entities of the configuration fragments. Moreover, we generate automatically the constraints corresponding to these semantic relations to complete the target system configuration profile. These constraints can be used to guard the configuration consistency during runtime modifications.

Keywords Configuration fragments · Model integration · Model weaving · Consistency · Constraints · Availability

1 Introduction

In order to handle complexity, large computing systems are generally built using the principle of separation of concerns. Different services/aspects are considered separately before integration. For instance, software and hardware aspects of a computing system may be considered separately before integration, or the functional, availability, performance or security aspects can be considered separately before integration. The separation of concerns principle eases the development process and increases reusability even if the integration may pose its own challenges. As the different aspects/services are considered separately their respective configurations are also developed separately. These configurations, used for management purposes, need to be integrated as well into a consistent system configuration in order to avoid conflicting management actions or actions from one aspect that may lead to malfunctioning of other aspects. The complexity of this integration stems from the potential overlap between the entities of the different aspect configurations (i.e. different logical representations of the same entity) and from the complex relationships among the entities of these different configurations, also referred to in this paper as configuration fragments. The integrated system configuration should reflect properly the relations and constraints between the entities of the different fragments and ensures that the resulting system meets the required properties of the different aspects in question, like availability, performance, security, etc.

We tackle the problem of integration of configuration fragments with a model driven approach based on the concept of model weaving (Del Fabro et al. 2005a). Model weaving allows for relating different models – in our case representing configuration fragments and referred to as configuration fragment models – by defining links between their entities. These links form a weaving model which conforms to a weaving metamodel. Model weaving has been widely used for model integration, model transformation, model merging, etc. (Reiter et al. 2005; Bernstein 2003; Jossic et al. 2007; Del Fabro and Valduriez 2007). However the focus so far has been primarily

This paper is an extension of reference (Jahanbanifar et al. 2015b) presented at ECMFA'2015.

✉ Ferhat Khendek
khendek@ece.concordia.ca

[1] Concordia University, Montreal, Canada

[2] Ericsson Inc., Montreal, Canada

on the static mapping of entities without considering the semantics of these relations. In our approach we take into account the semantics of these relations and target some desired properties of the resulting system configuration model. Our approach generates a consistent system configuration model which contains all the entities from the configuration fragment models, the constraints of each configuration fragment as well as the constraints reflecting the desired properties of the integration. The latter are generated automatically to capture the targeted properties entailed by the weaving links.

We use the Service Availability Forum (SA Forum) middleware (SAF 2010a) as a running example in describing our approach. The SA Forum middleware has been developed by a consortium of telecommunication and computing companies to support the development of Highly Available (HA) systems from Commercial-Off-The-Shelf (COTS) components. It consists of several services and frameworks, which represent and control specific aspects of the system and collaborate with each other (SAF 2010a; Toeroe and Tam 2012). Many of these services and frameworks that we refer to as services in the rest of this paper, require a configuration that specifies the organization and the characteristics of the system and/or application resources under their control. We focus on the configurations of two SA Forum services: the Availability Management Framework (AMF)(SAF 2010b), which manages the redundant software entities for maintaining the availability of application services, and the Platform Management (PLM) service (SAF 2010c) which is responsible for providing a logical view of the platform entities (hardware and low level software entities) of the system. Thus, they represent different aspects of a system and should be considered as configuration fragments. The configurations for these services are described using UML profiles. We capture the structure and the semantics of the relations between these profiles in a weaving model, which is later used to generate the system configuration including the constraints reflecting the targeted properties of the integration. Defining the relations between the profiles at a higher level of abstraction through a weaving model has several advantages such as reusability of the link types, increasing the extensibility (by allowing more models to be added) and automating the integration process (Del Fabro et al. 2005a, 2005b).

The rest of this paper is organized into six sections. In Section 2 we describe the example used throughout the paper. We discuss our approach for integrating configuration fragments (configuration models) in Section 3 while in Section 4 we focus on the automated generation of the constraints that reflect the relations between entities from the different fragments. In Section 5 we discuss the implementation of our approach, its scalability, extendibility and usability. Related work is reviewed in Section 6 before concluding in Section 7.

2 Running example

In this section we introduce a motivating scenario, also used as a running example to illustrate our approach for the integration of configuration fragments and constraint generation. As mentioned in the introduction, the SA Forum middleware consists of several services and frameworks and many of these services require a configuration for managing any application (resources) under the control of the middleware. These configuration fragments – generally developed separately – need to be integrated in a consistent manner while targeting certain properties, such as the high availability of the services provided by the application. In this paper we focus on the AMF and PLM services, their respective configurations, the relations between these configuration fragments and their integration.

2.1 Availability management framework (AMF)

AMF is responsible for maintaining the availability of application services by managing and coordinating the redundant software entities that compose this managed application (SAF 2010b). This management is based on the AMF configuration of the application, which describes the organization and the characteristics of the resources composing this application. A simplified example of an AMF configuration of an application is shown on the left side of Fig. 1. In an AMF configuration the service provider entities are called Service Units (SUs). The workload provisioned by an SU is represented as a Service Instance (SI). A group of redundant SUs providing and protecting the same SIs forms a Service Group (SG). An application may consist of a number of SGs. At runtime to provide and protect an SI, AMF assigns it in the active and standby roles to the SUs of the SG. In case of the failure of the SU with the active assignment AMF dynamically moves the active assignment from the faulty SU to the standby. Each SU is instantiated on an AMF Node, which is a logical container of SUs. SUs and SGs can be configured to be hosted on a particular group of AMF nodes referred to as a Node Group (NG). This means that such an SU/SG (the SUs of the SG) can be instantiated only on the Nodes of that Node Group (Toeroe and Tam 2012; SAF 2010b). An AMF configuration consists of these entities, their types and their attributes.

A complete description of AMF configurations is out of the scope of this paper, we only introduced the elements required for the understanding of the rest of the paper. More information on AMF configurations can be found in (SAF 2010b).

The concepts in an AMF configuration, their relationships, and the related constraints have been captured in an AMF configuration metamodel. A portion of this metamodel is shown on the right side of Fig.1. Subsequently, an AMF UML profile has been defined by mapping the AMF configuration metamodel to the UML metamodel (OMG 2011). The complete definitions of the AMF configuration metamodel

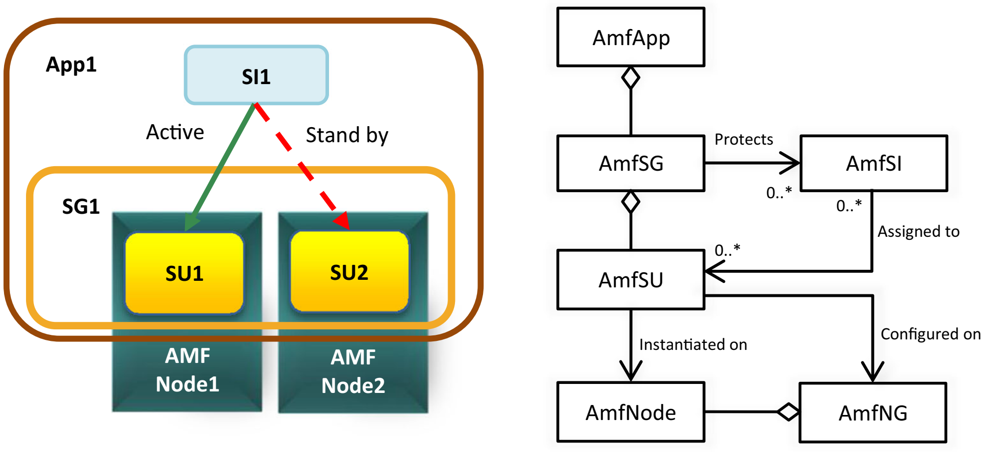


Fig. 1 A simplified AMF configuration and a portion of the AMF configuration metamodel

and the respective AMF UML profile are discussed in (Salehi et al. 2010). In our configuration integration approach we use this AMF UML profile as input.

2.2 Platform management (PLM)

The PLM service is responsible for providing a logical view of the platform entities of the system, which includes the Hardware Elements (HEs) and the low level software entities also known as the Execution Environments (EEs) (SAF 2010c). A PLM configuration represents their logical view. The PLM service manages the platform entities. A simple example of a PLM configuration is shown on the left side of Fig. 2.

In a PLM configuration PLM EEs represent software environments that can execute other software. A PLM EE can be an Operating System (OS), a Virtual Machine Monitor (VMM) i.e. a hypervisor or a Virtual Machine (VM) (Toeroe and Tam 2012; SAF 2010c). A PLM HE with computational capabilities can host a VMM or an OS. An OS can be parent of other PLM EEs, i.e. VMMs, and VMs can be hosted on VMMs.

As for the AMF we have defined a PLM metamodel, which captures the PLM configuration concepts, their relationships and their constraints. A portion of the PLM configuration metamodel is shown on the right side of Fig. 2. The PLM metamodel is based on the PLM specification in (SAF 2010c)., but further refines the standard PLM concepts and their relationships. For instance, the PlmEE is specialized into PlmEEVM, PlmEEVMM, and PlmEEOS. The relationship among these concepts has also been redefined: The relationship between the PlmEEVM and the PlmEEVMM is now defined through the PlmDependency. On the left side in the sample PLM model we use dashed lines between the VMs and their current hosting VMM, which is one of the VMMs listed in the dependency (not shown in the figure). The PlmEEVM has an association with its PlmEEOS. The PlmEEOS may have an association with a PlmEEVMM, i.e. host it.

These refinements are required to handle appropriately virtualized environments and cloud architectures. Multiple layers of PlmHEs may exist in a PLM configuration, e.g. in the configuration of Fig.2 there are HEHosts which are hosting the VMMs and the host OS while these Hosts themselves reside on HERacks. Following the same approach as for the AMF

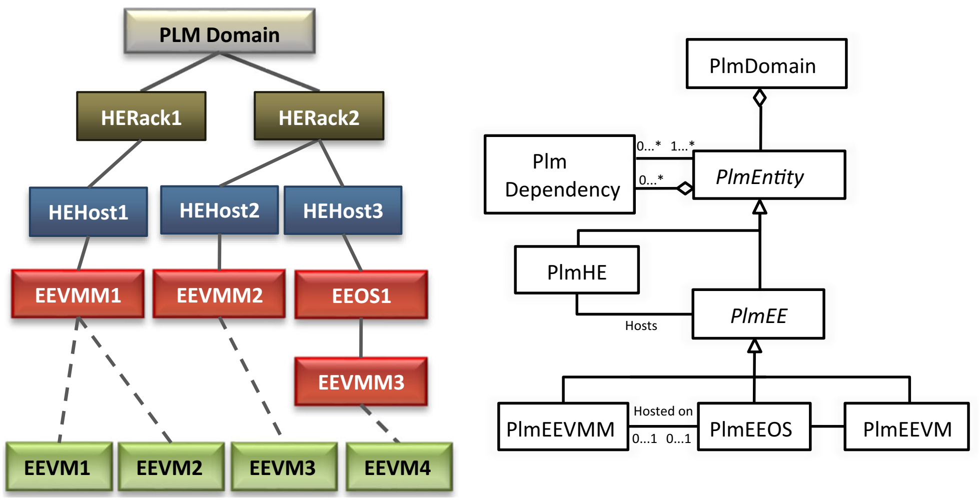


Fig. 2 A simplified PLM configuration and a portion of the PLM configuration metamodel

UML profile, we defined the PLM UML profile by mapping the PLM configuration metamodel to the UML metaclasses, with the closest semantics.

2.3 The integration relations among the fragments

According to the SA Forum specifications (SAF 2010b; SAF 2010c), each AMF Node is eventually hosted on (mapped to) a PlmEE so that the software entities of the AmfNode can be executed and provide services. This is basically the connection point between the two configuration fragments. In our work we assume this PlmEE is an OS instance installed on a VM instance. Therefore, an AMF Node is mapped to a PlmEEVM, and this is how the two configurations are put into relation. The question is whether any mapping between the AMF Nodes and the PLM EEs is acceptable?

We hereafter address this question through some examples explaining the specific property of the system that should be targeted in the integration of configuration fragments.

2.3.1 Hardware disjointness of service providers for enabling availability

Figure 3 shows a simple example in which the AMF configuration from Fig. 1 is put into relation with the PLM configuration of Fig. 2 by mapping AMFNode1 and AMFNode2 to EEVM2 and EEVM1, respectively. These two VMs are running on the same VMM and PLM HE (HEHost1). At this point the PLM HE as well as the VMM represent single points of failure. If this HEHost1 crashes both service providers, SU1 and SU2 will be lost and a service outage will be inevitable. Even if in the initial PLM configuration the VMs (EEVM1 and EEVM2) are hosted on different EEVMMs, at runtime the VMs may migrate and end up on the same VMM and HE at the same time. So, if the goal is to avoid any single point of failure due to the hosting hardware, we need to make sure that the service providers (SUs) of an SG will never be hosted on the same host.

2.3.2 Hardware affinity of service providers for fast communication

As mentioned earlier in Section 2.1 AMF manages redundant service providers (SUs) to avoid service outage due to SU failure. When the SU with the active assignment fails, AMF shifts the active assignment to the standby SU. To be able to use the standby SUs, the state of the active and the standby providers need to be synchronized so that in case of service failure the assignment of the service can be shifted without any service interruption. The active and standby SUs of an SG need to synchronize continuously and this state synchronization introduces some communication overhead causing latency in the normal behavior. The latency increases when the hosts of the SUs are farther from each other. E.g. in Fig. 4 SU1 is eventually hosted on HEHost2 and HERack2 while SU2 is eventually hosted on HEHost1 and HERack1. As the two SUs are residing on different HERacks, the latency is higher compared to the configuration in which the SUs are on the same HERack. Therefore, to assure an efficient communication (state synchronization) among the SUs of an SG and reduce this latency, the SUs should be placed closely together.

2.3.3 A combination of hardware availability and affinity

Each of the previous examples (hardware disjointness or affinity) shows an example of a property that may be targeted by a particular approach of integration of the configuration fragments. Moreover, more complex properties may be required such as the conjunction of both hardware affinity and disjointness, i.e. the SUs should be hosted on different hosts but the hosts should also

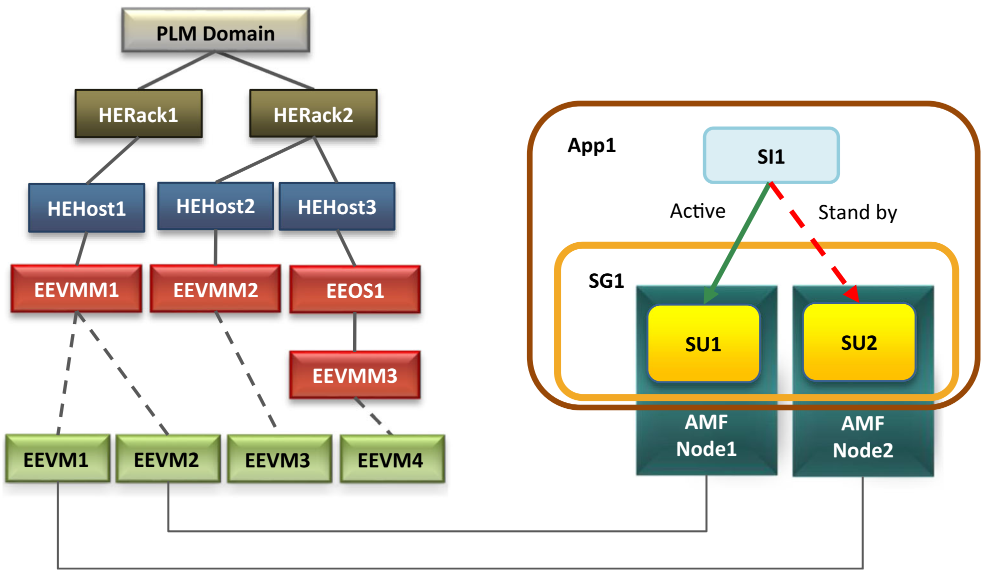


Fig. 3 Host failure problem because of the relation between the AMF and PLM configuration fragments

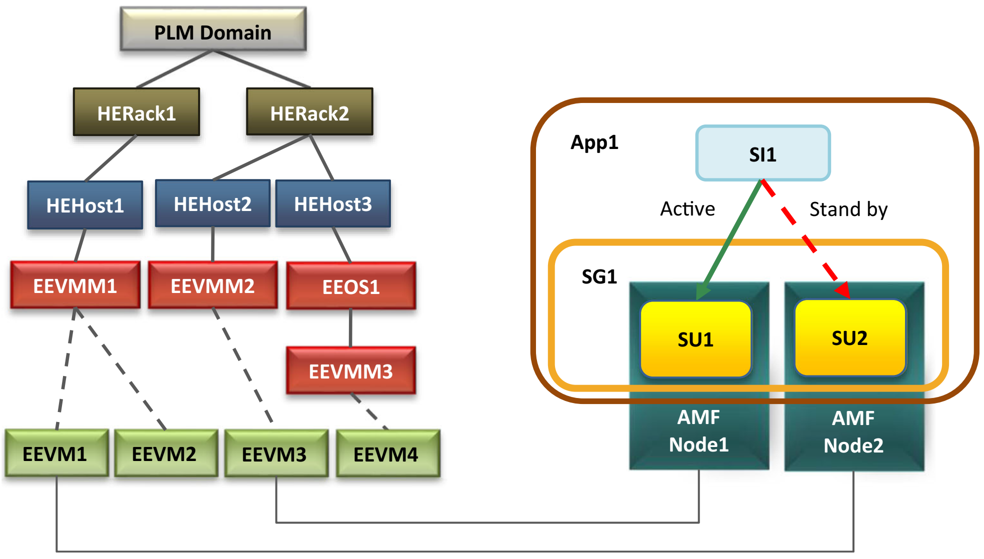


Fig. 4 Latency problem because of the relation between the AMF and PLM configuration fragments

keep certain proximity such as being in the same rack or site to assure the fast synchronization among the redundant SUs.

These examples of relations between the entities of the configuration fragments are examples of consistency rules which need to be captured and taken into account at the integration of the fragments. Moreover, these targeted relations/consistency rules will become constraints that will guard the consistency of the system configuration against runtime modifications.

3 The integration of configuration fragments

In this section we first discuss the challenges of the integration before introducing our model based integration approach.

3.1 The challenges

3.1.1 Overlapping entities

A configuration fragment is a logical representation of the resources for the management of their organization. A resource may exist in multiple configuration fragments with different logical representations. An example of a resource with multiple representations is a Virtual Machine (VM). As shown in Fig. 5 a VM is represented in the AMF configuration as an AMF Node and the same VM in the PLM configuration is represented as an EEVM.

Managing or modifying the overlapping entities (e.g. the entities with multiple logical representations) independently in each configuration fragment will lead to inconsistency in the system as they all affect the same resource. Thus, these logical representations of the same entity need to be related.

3.1.2 Integration relations between configuration fragments

The integration of configuration fragments usually targets certain properties for the system configuration. These properties depend and may involve more than one aspect of the system and thus require the capturing and the description of the required relations between these aspects. The hardware disjointness and affinity relations discussed previously are such examples. They need to be described

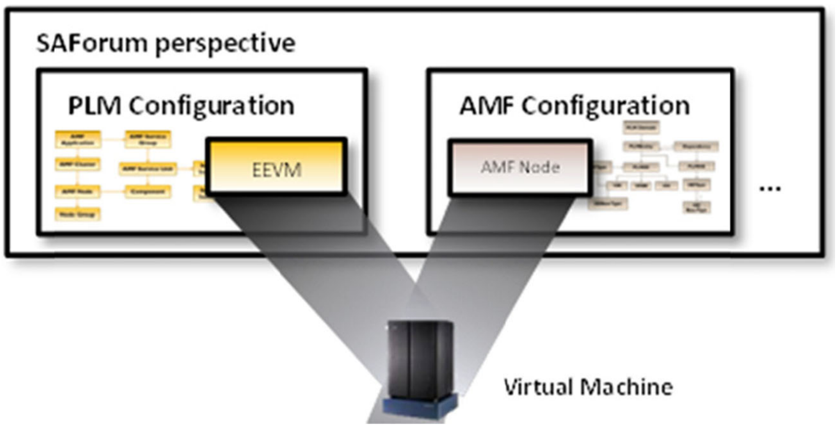


Fig. 5 Different representations of a Virtual Machine in different configuration fragments

and enforced by the integration to ensure properties like availability or lower latency for the system.

The relations between the configuration fragments need to be defined properly and according to the required properties. These relations should be enforced by the integration to define a consistent system configuration that exhibits the targeted properties.

3.2 The overall approach

To integrate configuration fragments we extend the model weaving technique. In this technique a model called the *weaving model* is used to capture the mappings between the entities of the metamodels. As any model in the model driven paradigm the weaving model conforms to a metamodel, i.e. the *weaving metamodel.* The weaving metamodel describes the types of the mappings that can be used in the weaving model. It also describes the types of entities which can be connected through these mapping types, i.e. the link end types. The instances of the mapping types (or link types) are used in the weaving model to connect the models'/metamodels' entities. The weaving model can have different applications such as defining traceability, tool interoperability, model transformation, etc.

As discussed earlier, for the integration of configuration fragments we need to capture more complicated relations among the fragments than just the entity mappings. Therefore, we extend the weaving concept in order to capture the semantics of the relations among the configuration fragment entities and use this semantics for the integration of the fragment models.

In our integration approach the configuration fragments and their metamodels are represented as the source models and source metamodels. For example, the AMF and PLM configuration models are the source models and their UML profiles are the source metamodels. We also use a system configuration metamodel that is called the target metamodel and at this stage it is a union of the source metamodels without any relationship between them. Through the weaving we integrate the source models and generate a system configuration, i.e. the target model.

We extend the weaving metamodel with special link types and we create a weaving model by defining the links between the configuration entities of the source and target metamodels. The weaving model is a static representation of the relations among the entities, therefore it is translated to an executable format using a Higher Order Transformation (HOT) (Tisi et al. 2009). The result of the HOT transformation is another transformation called *Final Transformation* which takes the source configuration models (e.g. the AMF and PLM configuration models) as the input and generates the target configuration model (i.e. the system configuration model) as the output. The overall process of the configuration integration through model weaving is shown in Fig. 6. In the following we summarize this process.

3.3 Extending the generic weaving metamodel

As mentioned earlier a weaving metamodel defines the link types and the link end types that can be used in the weaving model. Figure 7 shows part of a generic weaving metamodel (Del Fabro et al. 2005a, 2006) represented with lighter color elements. We extended this metamodel in order to capture the special relations between the configuration fragments. The elements extending the metamodel are shown in darker color in Fig. 7. In the following we explain these extensions in more details.

3.3.1 WLinkEnd specializations

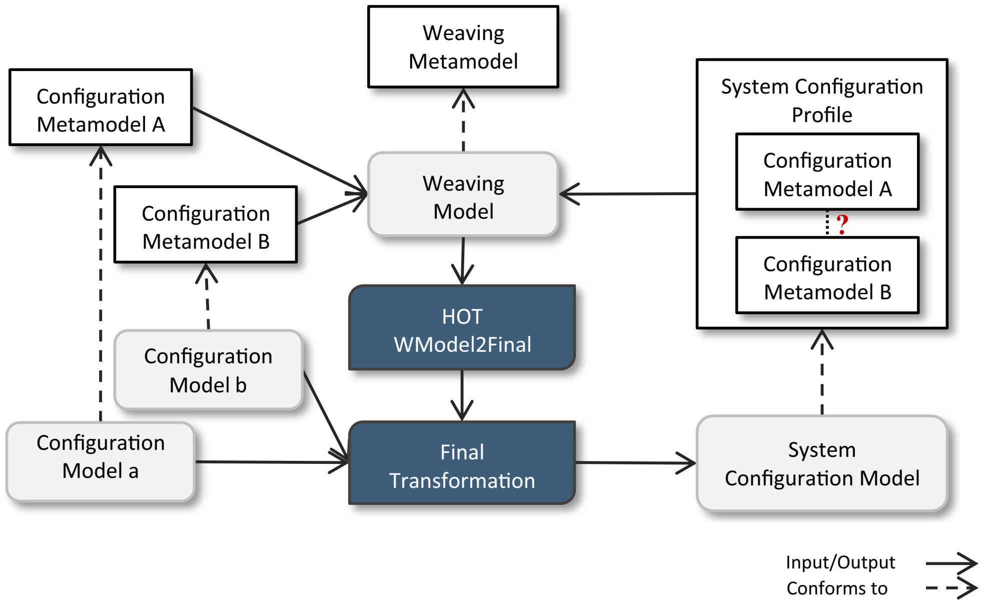
SourceEnd and TargetEnd In the generic weaving metamodel the WLink represents the generic link type which maps the WLinkEnds. For configuration integration we need to add a direction to the links and distinguish the source and target ends of the links as we have source models as input and we want to create the target model as output. Therefore, we consider the WLinkEnd as an abstract class and specialize it into the *SourceEnd* which is used to represent the configuration entities from the source models and the *TargetEnd* to represent the created/modified configuration entities which will appear in the target model (i.e. the system configuration). To make sure that in each link we have at least one SourceEnd and one TargetEnd constraint C1 is defined on the WLink. This constraint is expressed in OCL as:

```
Context WLink
Inv C1: Self.end->exists (e1, e2: WLinkEnd |
e1.oclIsKindOf (SourceEnd) AND
e2.oclIsKindOf (TargetEnd))
```

The entities of the source configuration metamodels that are specified as the SourceEnd are called the Source entities. The entities of the target metamodel appear in the TargetEnd and are called the Target entities. They are linked to the Source entities by the WLink.

Leader and peer The SourceEnd is specialized further into *Leader* and *Peer* link ends in the weaving metamodel to capture the influence of the configuration entities on each other. More specifically when a configuration entity is specified as a Peer and it is linked to a Target entity it means that the Target entity is created/modified with respect to the Peer Source entity (or Peer entity for short) however if the Peer entity also appears in the target model (i.e. created through the same or another link), these entities (the Peer and the Target entities)

Fig. 6 The system configuration generation through model weaving


Weaving Metamodel
Configuration Metamodel A
Configuration Metamodel B
Weaving Model
System Configuration Profile
Configuration Metamodel A
?
Configuration Metamodel B
HOT WModel2Final
Configuration Model b
Configuration Model a
Final Transformation
System Configuration Model
Input/Output
Conforms to

would have equal influence on each other in the target model. In the other words if either of them changes later in the target model, it can impact the other one.

Similar to the Peer link end type, the Leader link end is another specialization of the SourceEnd. Configuration entities specified as the Leader Source entities (Leader entities for short) also create/modify the Target entities but in contrast to the Peer entities, if the Leader entities appear in the target model (i.e. created through other links), only the Leader entities can influence the Target entities in the target model and not the other way around. This means that later in the target model if the Leader entities change, this change impacts their created/modified Target entities but if those Target entities change, they cannot impact the Leader entities.

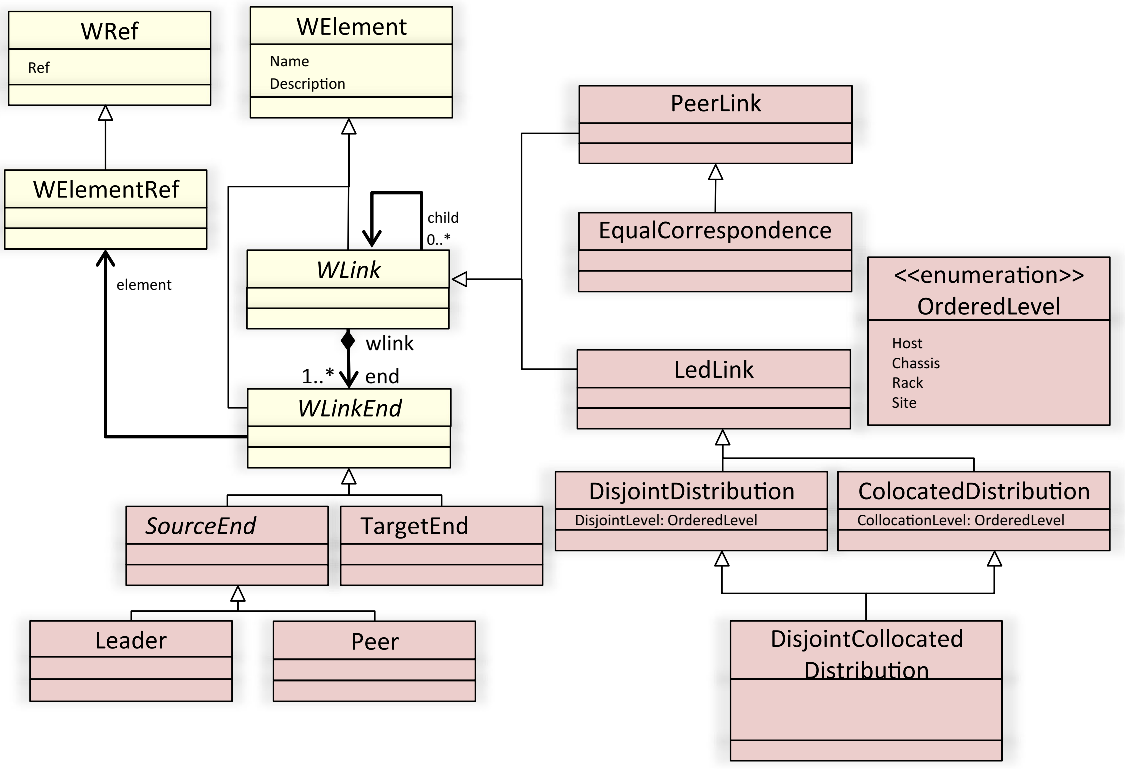


Fig. 7 The generic weaving metamodel extended with new LinkTypes and LinkEnds

3.3.2 WLink specializations

PeerLink The *PeerLink* represents the relation of the Peer Source entities and their Target entities. Defining a PeerLink among the Peer and Target entities means that even though the Peer entities are used to create/modify the Target entities the relation is not unidirectional. In the target model the relation is bidirectional, that is, the Target entities can have equal impact on the Source entities and vice versa. They are all in a Peer relation with respect to the constraints implied by the creation/modification rule.

A structural constraint, C2 is defined for the PeerLink to assure that the PeerLink has only Peer link end as its SourceEnd.

```
Context PeerLink
Inv C2: Self.end->forAll (e: WLinkEnd |
e.oclIsKindOf(SourceEnd) implies
e.oclIsTypeOf(Peer))
```

EqualCorrespondence In the configuration integration it happens that many entities from the source models are just copied to the target model. The *EqualCorrespondence* link type, inspired by (Jossic et al. 2007), is defined to map the Source entities to their identical Target entities. EqualCorrespondence is a specialization of the PeerLink so the Peer link end is used as the SourceEnd for this link type and the TargetEnd is the other link end for this link type.

LedLink The *LedLink* represents the relation of the Leader Source entities and the Target entities. It means that when a LedLink is defined among the Leader and Target entities, the Leader entities create/modify the Target entities and such impact or affection among the entities (i.e. Leader entities impact the Target entities) needs to be maintained in the target model among the involved entities.

A structural constraint, C3 is defined for the LedLink to assure that the LedLink has only Leader link end as its SourceEnd. This constraint is expressed in OCL as:

```
Context LedLink
Inv C3: Self.end->forAll (e: WLinkEnd |
e.oclIsKindOf(SourceEnd) implies
e.oclIsTypeOf(Leader))
```

DisjointDistribution The *DisjointDistribution* link type is defined to capture the hardware-disjointness property for the target configuration. DisjointDistribution is an extension of the LedLink and therefore the Leader link end and also the TargetEnd needs to be specified for the link. This link type has an attribute called *DisjointLevel* of an enumeration type *OrderedLevel.* The OrderedLevel enumeration has the items of Host, Chassis, Rack, Site, and Geographic which define the levels of disjointness that might be required for the configuration entities. E.g. for the scenario we explained earlier, if the DisjointLevel attribute is set to Host, then the linked entities should be configured on different Hosts. If this attribute is set to Rack, for instance, the linked entities must be configured for different Racks. The values of the OrderedLevel type are defined according to the Open Virtualization Format specification (DMTF 2013).

CollocatedDistribution The *CollocatedDistribution* link type is defined similarly to the DisjointDistribution but with another purpose; to capture the collocation requirement in the relations between the entities of the fragments. CollocatedDistributaion is also specialized from LedLink and has a *CollocationLevel* attribute. This link guarantees that the target entities are configured for groups of collocated source entities. For example, in the usecase of Section 2 if the SUs are required to be configured on the HEs of the same Rack, the CollocationLevel is set to the required level, i.e. Rack.

DisjointCollocatedDistribution In Section 2 we mentioned that both the availability and affinity of the service providers may be required. However, these properties can be conflicting and should not be considered independently if both are required. To capture such relation another link type is added which inherits from both CollocatedDistribution and DisjointDistribution and thus has the properties of both. To make sure that the two concepts are not introducing any conflict, we make sure that each concept is applied in a different level. This means that the level of providing availability through DisjointDistribution should be different from the affinity level provided by the CollocatedDistribution. The DisjointLevel and CorelationLevel attributes allow us to make such distinction. However, the levels cannot be selected arbitrarily and need to respect a rule. To define this rule we again followed the OVF specification (DMTF 2013) which indicates that the collocation property should be provided in a higher level than the disjointness. This means that for example if the disjointness is provided at Host level, then the collocation level can be Chassis, Rack, Site or Geographic. This rule can be specified with an OCL constraint in the weaving metamodel as follows:

```
Context DisjointCollocatedDistribution
Inv C4: OrderedLevel.allInstances()- >
indexOf(self .DisjointLevel) < OrderedLevel.
allInstances()- > indexOf (self.
CollocationLevel)
```

3.4 Creating the links in the weaving model

Once the required link types have been defined in the weaving metamodel, they can be used in the weaving model for relating entities of the source metamodels to the entities of the target (system configuration) metamodel. The weaving model includes instances of links (instances of link types) associated with their respective link ends. Examples of these links are described in the following for EqualCorrespondence and DisjointDistribution link types.

The DisjointDistribution link type of the weaving metamodel is used to represent the disjointness relation between the relevant entities.

In the case of the integration of the AMF and PLM configurations, if we assume a fixed hardware platform and accordingly the PLM configuration is fixed and cannot be changed as part of the integration, then the AMF entities (i.e. the Nodes, NGs and SUs) should be configured according to the entities of the relevant PLM configuration (i.e. VMs and HEs) to satisfy the hardware disjointness constraint. Thus, in the DisjointDistribution link the PlmEEVM and PlmHE entities of the PLM configuration metamodel are the Leader SourceEnd and the AmfNode, AmfNG, and AmfSU are the TargetEnds. The application of this link type with Host disjointness is as follows:

```
<<WLink>> DisjointDistribution HEDisjointSUs
<DisjointLevel>
    OrderedLevel    Host
<Source>
    <<Leader>>        PlmEEVM
    <<Leader>>        PlmHE
<Target>
    <<TargetEnd>>   AmfNode
    <<TargetEnd>>   AmfNG
    <<TargetEnd>>   AmfSU
```

In more details this link indicates that the AmfNode, AmfNGs and AmfSU entities in the target model are created or modified with respect to the PlmEEVM and PlmHEHost. These creations/modifications should happen in such a way that Host disjoint is provided for the AmfSUs. The CollocatedDistribution is used in a similar manner. We are not going to discuss it in details.

An instance of the EqualCorrespondence link type is used to map an entity of a source metamodel to a similar entity of the target metamodel. Some semi-automated methods such as the technique introduced in (Del Fabro and Valduriez 2007) can be applied to automate the creation of the mappings based on the similarity (such as string or type similarity) of the entities. Such automation can be applied only after all other types of links have been defined in the weaving model.

```
<<WLink>> EqualCorrespondence EqualVMs
  <Source>   <<Peer>>        PlmEEVM
  <Target>   <<TargetEnd>>   SystemEEVM
```

In the next section we explain how the links are translated to transformation rules to create the target model with respect to the semantics of the relations (links).

3.5 Generating the system configuration from the weaving model

To be able to generate a system configuration model it is necessary to translate the weaving model into an executable format. This translation takes place using an HOT, which itself is a transformation. The HOT translates the links of the weaving model into transformation rules. The output of the HOT is the Final Transformation as shown in Fig. 6. An excerpt of the HOT for creating the module section in ATL is shown below.

```
create OUT: ATL from IN: AMW, sourceModel1:
MOF, sourceModel2: MOF, targetModel: MOF ;

rule ModuleCreation{
     from
          amw: AMW!Model
     to
          atl: ATL!Module(
             isRefining <- false,
             name <- 'ModelTransform' ,
             inModels<-Set{ amw.source
             Model1,amw.sourceModel2} ,
             outModels <- amw.targetModel,
             elements <- Set{ amw.Wlink} ,)}
```

The *inModels* specifies the input metamodels for the generated transformation while the *outModel* indicates the output metamodel of the final transformation (amw.target that is the UML profile for system configuration). The *elements* of the final transformation is created by transforming the set of Wlinks of the weaving model (*Set{amw.Wlink}*). Each link type and its linkEnds of the Wmodel are translated to the elements of the transformation model with other HOT rules.

The translation of the DisjointDistribution link, for instance, results in several transformation rules (expressed in ATL) in the Final Transformation. This translation is done with respect to the algorithm which we introduced previously in (Jahanbanifar et al. 2014) to create hardware disjoint groups of VMs and the respective AmfNodeGroups to configure the AmfSUs on the AmfNodeGroups. The high level overview of these ATL rules and a brief description of each are provided hereafter:

```
rule NodeVM_AssociationCreation(id:
Sequence(Integer))
rule VMG_Creation()
rule NG_Creation(vmg: Sequence(OclAny))
rule SUNG_AssociationCreation(su:AMF!AmfSU,
index:Integer)
```

The NodeVM_AssociationCreation rule creates a relation (an association) between each distinct pair of PlmEEVM and AmfNode, e.g. associating an AmfNode to the most similar

PlmEEVM regarding the capacity of the two entities. The association of a PlmEEVM to an AmfNode entity can be seen as an attribute of the AmfNode in the target model. This relation is the basic connection between the entities of the two configuration models.

```
rule Node_VM_Association_Creation(id:
Sequence(Integer)){
  to
    target: System!Association (memberEnd<
    - Set{ p1,p2} ),
    p1: System!Property(
        name <- 'src' ,
        type <-AMF!Node.allInstances
        From('IN1')-> select(
          c|c.hasStereotype('AMF::AmfNo-
          de'))-> select(s|s.ID=id)
                      )
    p2: System!Property(
        name<- 'dst' ,
        type<- PLM!Node.allInstances
        From('IN2')->select(
          c|c.hasStereotype('PLM::PlmEE-
          VM'))-> select(s|s.ID=id)
                      ) }
```

The VMG_Creation rule is used to generate the Host hardware disjoint VM Groups (VMGs) based on the input PLM configuration model according to Algorithm 1 introduced in (Jahanbanifar et al. 2014). The isolation of this calculation in a rule makes its modification or replacement by another algorithm easy and avoids touching the rest of the transformation model. This rule creates VMGs (each of which is a *sequence* of VMs) collected in a VMGSet (which is a *sequence* of *sequences* in ATL). The VMGSet and its VMGs do not appear in the target configuration but are used to create an NGSet and its NGs.

The NG_Creation rule creates the AmfNGs in the target model based on the previously created VMGs of the VMGSet and adds the relevant AmfNode entities to the created the AmfNG. In our translation we assume that we do not have the AmfNG entities in the AMF model, so we create them in the target model. Alternatively if the AMF model contains the AmfNGs, we would need to re-configure them instead of creating new ones, which may be limited by additional constraints.

```
rule NG_Creation(vmg: Sequence(OclAny)){
  to
    target:System!Class(
        name <-' NG' + thisModule.VMGSet->
        indexOf(vmg) ),
    t1: System! AmfNG (base_Class <- target)
  do{
    thisModule.NGSet<-thisModule.NGSet->
    union(Sequence{ target} );
    for(vm in vmg){ thisModule.NGN_
    Association_Creation(vm,target)} ;
    }    }
```

Algorithm 1 Defining HE-disjoint groups of VMs

```
Input: HW_Dependency from the PLM configuration
Output: Set of HE-disjoint VGs in VMGSet and a set of unused VMs in
  leftovers
1: A: = set of all VMs in PLM configuration
2: Leftovers: = {}
3: VMGSet: = {}
4: // Identify the HEs related to each VMi based on HW_
   Dependency from the PLM configuration
5: for each VMi in A do
6:   HE-VMi: = {HEs related to VMi in HW_Dependency}
7: end for
8: // Select among remaining VMs the VM associated with the
   lowest number of HEs in HW_Dependency and remove from
   set A the VMs that are not HE-disjoint with it
9: // n is the counter of the VMGs
10: n: = 0
11: repeat
12: select VMi from A such that |HE-VMi| ≤ |HE-VMj| for any
    VMj in A
13:   n: = n + 1
14:   VMGn: = {VMi}
15:   HE-VMGn: = HE-VMi
16:   A: = A – {VMi}
17:   for each VMj in A do
18:     if HE-VMj = HE-VMGn then
19:       VMGn = VMGn ∪ {VMj}
20:         A = A – {VMj}
21:     else if HE-VMj ∩ HE-VMGn ≠ {} then
22:         Leftovers: = Leftovers ∪ {VMj}
23:         A = A – {VMj}
24:     end if
25:   end for
26: VMGSet: = VMGSet ∪ {VMGn}
27: until A = {}
28: //Adding the leftover VMs to formed VMGs iff they intersect
    with only one VMG with respect to their HEs
29: for each VMi in Leftovers do
30: // k is the counter of VMGs with which VMi has common HEs
31:   k: = 0
32: for each VMGj in VMGSet do
33:     if HE-VMi ∩ HE-VMGj ≠ {} then
34:       k++
35:       temp: = j
36:     end if
37: end for
38:   if k = 1 then
39:     VMGtemp = VMGtemp ∪ {VMi}
40:     HE-VMGtemps: = HE-VMGtemp ∪ HE-VMi
41:     Leftovers: = Leftovers – {VMi}
42:   end if
43: end for
44: return VMGSet
```

Finally, the SUNG_AssociationCreation rule is used to establish the relation (association) between the AmfSUs of each

AmfSG and an AmfNG entity which was created by the previous rule. In this work we do not consider any criteria for matching the AmfSUs to AmfNGs. This rule can be extended in the future by adding different heuristics for selecting the most appropriate AmfNG for each AmfSU based on some criteria (such as the number of AmfSUs in the AmfSG, etc..).

```
rule SUNG_AssociationCreation(su:AMF!AmfSU,
index:Integer){
    to
      target: System!Association
      (memberEnd<- Set{ p1,p2} ),
      t1: System!SUSG(base_Association <-
      target),
      p1: System!Property(
          name <- 'src' ,
          -accessing the created SU in the
          target model
          type<-thisModule.resolveTemp
          (su,' targetSide' )),
      p2: System!Property(
          name<- 'dst' ,
          type<-thisModule.NGSet->at
          (index) ) }
```

The Final Transformation generated from the HOT takes the configuration fragment models as input and generates a system configuration model as output. The generated configuration will have all the entities of both input models and also the new entities and relations among the entities of the fragments capturing the targeted properties entailed by the weaving links.

4 Constraint generation from the integration

The transformation rules in the Final Transformation are generated by considering the special relations among the entities of the configuration fragments. These relations guarantee the targeted properties of the system configuration, i.e. the consistency of the system configuration with respect to the targeted properties such as availability and affinity.

Although this integration semantics is taken care of in the process of generating the system configuration model, it is not reflected in the system configuration profile. This integration semantics needs to be defined as integration constraints in the system configuration profile in order to guard the consistency of system configuration models against unsafe runtime modifications. The integration constraints (i.e. originating from the transformation rules and describing the semantics of the relation between the fragments) in addition to the union of the constraints of the fragments form the system configuration constraints. The transformation rules of the Final Transformation can be reused to generate automatically the integration constraints. The configuration designer does not have to define them manually as they are already embedded in the transformation rules. In this section we describe how the integration constraints can be extracted. We describe our approach for the generation of OCL constraints from the ATL transformation rules. Figure 8 shows the constraint generation from the Final Transformation and the completion of the system configuration profile.

An ATL transformation model consists of rules and helpers. There are three types of transformation rules in ATL: matched rules, lazy rules and called rules. The most commonly used rule type is the matched rule, which generates the target entities from the source entities defined by the source pattern of the rule. A matched rule is executed for all the occurrences of its source pattern. In contrast to the matched rule, a lazy rule is executed only when it is invoked. Finally, called rules are used to create target entities from imperative code. To be executed the called rules need to be invoked from an imperative code, which can be the action block of a matched rule, or from within another called rule. Helpers in the context of ATL are similar to methods. The helpers can be called from different points of an ATL program.

4.1 Entity derivation tree

Each target entity or its attribute is created by some transformation rules and helpers. If the entity is created in a rule and its attributes are created in some other rules, we consider the attribute creation as an entity modification. By following the transformation rules and helper invocations for the creation/modification of each target entity we can specify the process of its creation/modification as a *derivation tree*. At the root of the tree there is a target entity and at each level of the tree the nodes are the entities which are used to create their parent node and the edges are the operations that are applied on the nodes to create the parent node. An operation can be a rule/helper invocation, a filter or guard expression, or a piece of imperative code in the rules. At the last level of the tree are the leaves (entities) which already exist in the target model or the source models. Although the derivation tree can be created for each target entity, we are interested only in the target entities that are created/modified using some operations.

Traversing this tree from the root to the leaves helps in identifying the entities and operations that are used to create/modify a target entity. On the other hand, exploring the tree from the leaves to the root describes how a target entity is created/modified from the operations, the effect of which needs to be captured as constraints between the entities of the system configuration.

Figure 9 shows a very simple example of a derivation tree for creating the SystemEEVM from the PlmEEVM. A simple transformation rule called VM_Transformation is used to copy the PlmEEVM entities from the PLM configuration fragments to the

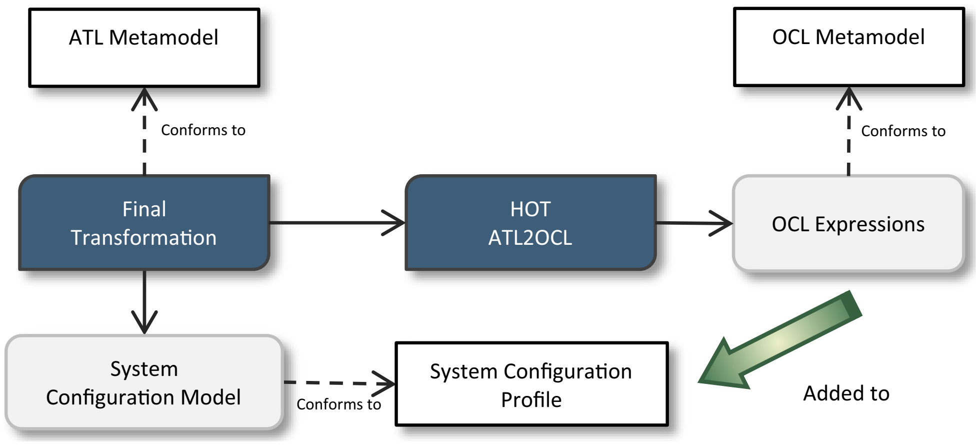


Fig. 8 Generation of OCL constraints from the ATL transformation

system configuration and create the SystemEEVM entities. This rule is the translation of the *EqualVMs* link in Section 3.4 (i.e. an EqualCorrespondence weaving link). Let assume that we want only the VMs with Memory of 512 MB or higher to be used for the target entity creation. Therefore the source pattern used in the VM_Transformation rule uses a filter on the entities of the source. Starting from the target entity and following the transformation rule creating it, we reach the source entity to which the filter operation was applied. This is shown as *Traversing direction* in Fig. 9. The target entity node (SystemEEVM) is created from the filtered source node (PlmEEVM) which is shown as *Entity creation direction* in the figure. The filter is an example of an operation that can be applied to source entities, it is shown on the edge connecting the nodes. Other operations can be helpers, called rules, or lazy rules.

```
rule VM_Transformation{
  from source:PLM!PlmEEVM(source.Memory>
       512)
  to    target: System! SystemEEVM(
        Memory<- source.Memory ) }
```

More examples of derivation trees are shown in Fig. 10, which is based on the DisjointDistribution link and its respective transformation rules. Figure 10 (a) shows the derivation tree of the modification of the AmfNode entity to map it to a VM in the PLM configuration fragment (the association between the AmfNode and PlmEEVM is considered as an attribute of the AmfNode). This tree has only one level and the operation on the edge between the root (AmfNode) and the leaf (PlmEEVM) is the rule NodeVM_AssociationCreation rule, i.e. a called rule which selects a distinct PlmEEVM for the AmfNode possibly based on some other criteria such as the capacity of the VM.

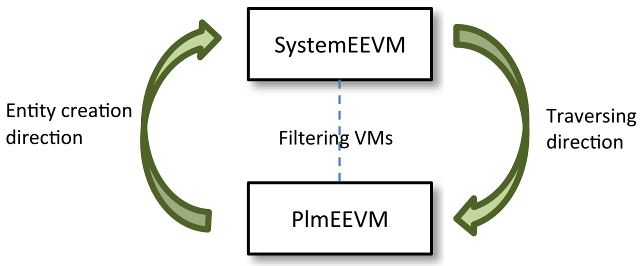


Fig. 9 An example of derivation tree for SystemEEVM

Figure 10(b) is the derivation tree for the creation of the AmfNG. This tree has two levels: level 1 includes the AmfNode and the VMG tree nodes on which the NG_creation operation (i.e. a called rule for creating NGs from the VMGs) was applied at this level. As the AmfNode exists in the system model, it is a leaf node of tree. On the other hand no VMG entity exists in the source or the target models. It is an entity which is only created and used in the transformation rules as an auxiliary entity. The VMG entity represented by the VMG node of the tree is created from the PlmHe, PlmDependency and PlmEEVM entities of the system model. To preserve any constraint implied by these operations in relation to the AmfNG, we need to include in our derivation tree these as well. Thus these entities are shown as the tree nodes in level 2 of the derivation tree. The VMG_Creation (i.e. a called rule) and the HE_Checking (i.e. a helper) are the operations applied on the nodes of level 2 to create their parent which is the VMG.

Figure 10(c) shows the derivation tree for the modification of the AmfSU entity (the association between the AmfSU and AmfNG is considered as an attribute of the AmfSU). This tree has only one level and the operation on the edge between the root (AmfSU) and the leaf (AmfNG) is the rule SUNG_AssociationCreation rule (i.e. a called rule which selects a distinct NG for the AmfSUs).

4.2 Translation of the ATL operations to the OCL expressions

Once a derivation tree is created, from the operations applied on the nodes of each level we need to derive an appropriate OCL expression.

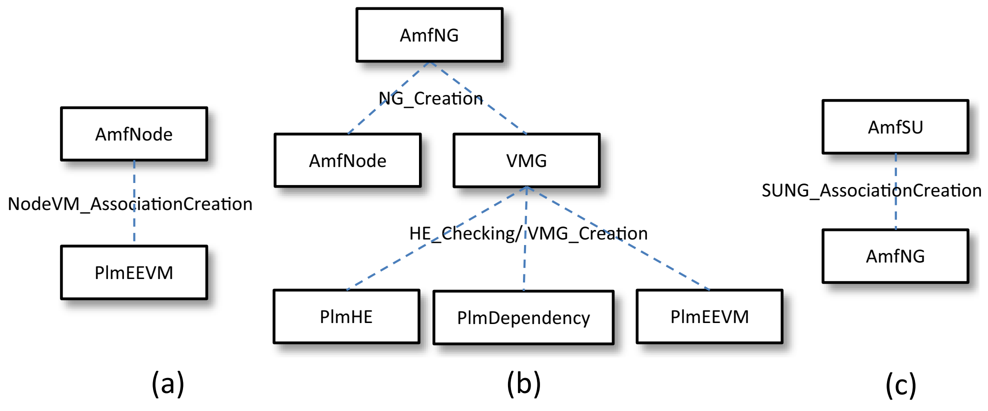


Fig. 10 The derivation trees for the AmfNode, AmfNG, and AmfSU

The context of a generated OCL expression at each level is the parent entity if this entity exists in the target model. E,g. for tree (a) of Fig. 10, AmfNode is the parent node and it is a target entity, which exists in the target model, therefore the context of the generated OCL expression from this tree is AmfNode.

However at any level of the tree if a parent does not exist in the target model (i.e. the parent is an auxiliary entity which is only used in the transformation) then the parent entity cannot be the context of the OCL expression generated for its subtree. (Note that this cannot happen to the root of the tree, which is a target entity and therefore it is always in the target model.) In such cases the context is the same as for the level above, e.g. the parent of this parent. An example of this case is the VMG entity in tree (b) of Fig. 10, which is created from the PlmHE, PlmDependency and PlmEEVM entities, but the VMG entity does not exist in the target model. So the context of the OCL expression created from the VMG_Creation and HE_Checking cannot be the VMG and is defined as for the level above, i.e. the parent of the VMG entity in the tree, which is the AmfNG.

To derive the OCL expression, we have categorized the ATL operations and the OCL expression respectively into types and define the types mapping. Table 1 summarizes the ATL operation types we identified for common ATL operations and the mappings of these ATL operation types to OCL expression types. These mappings are then defined as an HOT transformation (i.e. the ATL2OCL transformation of Fig. 8). Thus, we reuse the mappings for similar operations. The OCL expressions resulting from applying this mapping to the derivation trees of Fig. 10 are shown in Table 2.

4.3 Role definition for the constrained entities

In addition to the generation of the OCL expressions we can also capture the role of constrained entities in the constraints. We previously extended the OCL by defining Leader/Follower/Peer roles for the constrained entities to show the influence of the entities over each other (Jahanbanifar et al. 2015a). Figure 11 shows the extension of the constraints with the leadership information.

As we used a similar leadership concept in the weaving process, the Leader/Follower/Peer role of constrained entities in the integration constraints can be obtained automatically: The entities specified as the Leader SourceEnd of the LedLink take the Leader role and entities specified as the TargetEnd of the LedLink have the Follower role in the constraint generated from the LedLink and its transformation rules.

In the other link types, the SourceEnd is Peer and therefore, both the Peer Source entities and the Target entities of the link will have the Peer role in the LeadershipInfo of the generated constraint as they have equal influence over each other in the target model.

In the DisjointDistribution weaving link between the AMF and PLM configurations, the PlmEEVM and the PlmHE (the Leader Source entities) have an influence on the AmfNode, AmfNG and AmfSU (the Target entities). Accordingly in the generated constraint the PlmEEVM and the PlmHE entities

Table 1 The mapping of ATL operations to OCL expressions

ATL Operation Type	OCL Expression Type
Type operations in the filters (operations on primitive or collection types e.g. *select*, *iterate*, so on)	Type operations as the invariant of constraint
Matched rules with iterative binding of entities' attributes (e.g. *for* loop)	Defined by *allInstances* or *forAll* expressions
Variables in the *Using* section	Defined by *let* expression
Helpers, Lazy rules, Called rules	Defined as the Body of Query operations

Table 2 The OCL expressions resulting from the derivation trees of Fig. 10

Tree	ATL Operation	Operation Description	OCL Expression
a	NodeVM_ Association Creation	-It maps each VM to a distinct Node -It requires PlmEEVM	Context AmfNode Inv: self.allInstances- > forAll(N1,N2| N1 <> N2 implies N1.vm <>N2.vm)
b	**Applied on Level2:** VMG_ Creation, HE_Checking	-They are called to create the VMG -They require the PlmHE, the PlmDependency and the PlmEEVM	Context AmfNG::Disjointness(Ng1,Ng2):Boolean Body: If (Ng1.node - > iterate (N; VMM1: PlmEEVMM | VMM1- > including (N.vm.dependency.supplier))- > iterate (VMM; HE1: PlmHE | HE1- > including (VMM.he)) - > intersection(Ng2.node - > iterate (N; VMM2: PlmEEVMM |VMM2- > including (N.vm.dependency.supplier))- > iterate (VMM; HE2: PlmHE | HE2- > including (VMM.he)) - > isEmpty()) then return True else return False endIf
	Applied on Level1: NG_Creation	-It is called to create the NGs -It requires the VMG and the AmfNode	Context AmfNG Inv: self.allInstances- > forAll(Ng1,Ng2|Disjointness (Ng1,Ng2) = True)
c	SUNG_AssociationCreation	-It modifies the SUs. -It iterates over the SUs of each SG to associate each SU with a distinct NG	Context AmfSU Inv: self.allInstances- > forAll(Su1,Su2| Su1.sg = Su2.sg implies Su1.ng <>Su2.ng)

Note that for tree (b) Level 2 defines the Disjointness method in the context of the AmfNG, which is referenced at Level 1

have the Leader role and can affect the AmfNodes, AmfNGs and the AmfSUs which have the Follower role in the LeadershipInfo of the corresponding constraints.

It is worth mentioning that the roles of the constrained entities may change with the application scenario. More specifically we may define the Leader/Follower/Peer roles for the entities for the initial design time of a configuration to meet the setup and deployment requirements. However after the system is deployed and the requirements change we may be limited in the changes allowed. Therefore the roles of the entities in the constraint may change although the OCL constraint remains unchanged. Defining the roles for the entities through the LeadershipInfo has this advantage that we can define and change the roles whenever it is needed without affecting the constraints themselves.

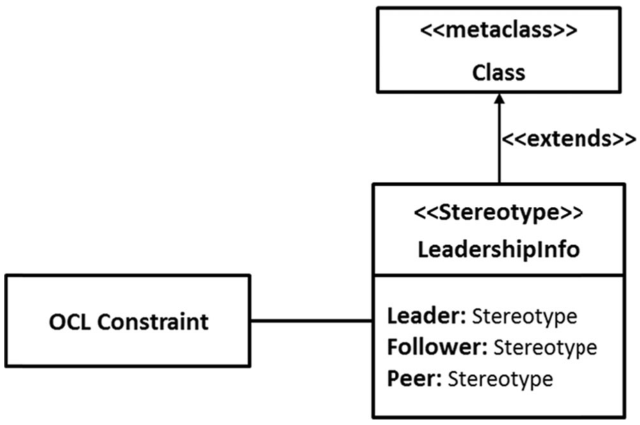


Fig. 11 OCL constraint enriched by the LeadershipInfo

The LeadershipInfo can also be defined for the other constraints of the system configuration (i.e. the constraints of the configuration fragments). Knowing the roles helps us to recognize the necessity and the order in which the constraints need to be checked as explained in (Jahanbanifar et al. 2015a).

5 Discussion and assessment

5.1 Implementation and scalability

We have implemented our approach using the Atlas Model Weaver (AMW) (Del Fabro et al. 2005b, 2006; Eclipse 2010), and the Eclipse Modeling Framework (EMF) (Eclipse 2014). We enriched the generic weaving metamodel of the AMW defined in KM3 (Jouault and Bézivin 2006) with the extensions discussed in Section 3. We used ATL (Jouault et al. 2008) as our model transformation language for the implementation of the transformations. The algorithm defined in (Jahanbanifar et al. 2014) is used as an instance for the implementation of the DisjointDistribution link, however it can be replaced with alternative algorithms.

We have worked with relatively small size configuration fragments to demonstrate the ideas and the concepts in our approach. The scalability of our approach in terms of model size is not an issue as demonstrated by the experiments conducted in (Del Fabro and Valduriez 2009) with AMW for model weaving and transformations generation. Indeed in (Del Fabro and Valduriez 2009) the authors varied their model sizes, in terms of elements, classes, attributes, from small to large and demonstrated the scalability of the model weaving and transformation generation techniques. The processing remains in terms of seconds.

Modeling configuration fragments separately and merging them to define the system configuration according to certain semantic relationships is a simple and powerful process along the lines of well-known software engineering principle of separation of concern.

5.2 Application domain

Although we explained our approach in the context of the SA Forum, we believe it is generally applicable to model integration in any domain. Our approach is based on the model weaving technique and focuses on the semantics of the relations in the weaving. Examples of such semantics are the HW-disjointness property (to ensure hardware redundancy for redundant software entities to increase the availability of the system), the HW-collocation property (to decrease the communication latency due to state synchronization) or the combination of the two. Our approach can be applied for instance in the automotive domain where several components are developed separately and integrated, according to some semantic relations, to form the system. (Nechypurenko et al. 2007) looked into the merging techniques in this domain to reduce the complexity of developing automotive systems from a different angle including the usage of constraint solvers.

5.3 Reusability and extensibility

Defining special link types in the weaving metamodel allows for the development of more abstract mappings. Abstracting concepts is an intrinsic feature of metamodeling, which is discussed widely in the literature. This advantage becomes bolder in the case of configuration integration from two perspectives: First it increases the reusability of link types since the defined link types can be used in future mappings when other configuration fragments need to be added. As those configuration fragments belong to the same system, there is a fair chance that they require similar link types for their mappings (e.g. using the "EqualCorrespondence" link type). The second advantage of the abstract definition of link types is that it allows for the selection of the desirable interpretation and implementation for the mapping. This means that the declarative definition of the link types can be translated according to the features of the system. Let us consider the "DisjointDistribution" link type. We interpreted this link with the assumption that we have a predefined PLM configuration with specific entities that are fixed and cannot be modified; on the other hand we forced the AMF to use the newly defined VM groups by changing the AMF configuration. While another interpretation of the "DisjointDistribution" may consider the AMF configuration as fixed and unchangeable model and using other heuristics try to change the PLM configuration in a way to still provide hardware redundancy for redundant software entities.

The fact that model transformation is used to translate the weaving model into an executable format does not fade away the benefits of the weaving model. The reason is that the links of the weaving model help us to capture the transformation patterns and reuse them rather than defining all the rules manually. Another reason for selecting the model weaving technique over the direct model transformation is the extensibility of the weaving for integrating additional models with less manual effort. With model weaving we can simply add more models as input into the weaving process and the respective transformation rules will be generated automatically, while adding more models directly into a transformation requires considerable time and effort to develop the new transformations rules.

5.4 Usability

Our approach as illustrated in Figs. 6 and 8 requires metamodels, models, and transformation. For a given domain, like SA Forum, one has to come up first with the configuration metamodels. This is usually achieved by a domain expert knowledgeable in MDD or with the help of MDD experts. Extending the weaving metamodel requires similar expertise. These metamodels are designed only once and used as many times as needed. A configuration fragment model is an instance of one of these metamodels, creating such an instance is not a complex task for a user with some modeling knowledge. In our case it is basically coming up with a class diagram that conforms to the metamodel. A domain expert can design the weaving model that captures the semantics of the integration at a high level of abstraction. This is an advantage which reduces the risks of errors; however this model is not executable and one has to come up with a transformation (Final Transformation) to realize this semantic weaving of the input models. Therefore, the weaving model has to be translated into an ATL transformation, and this is achieved with a high order transformation (WModel2Final HOT) written once by a modeling expert. Once the metamodels and the weaving models are created and the ATL transformation is generated one can integrate as many configuration fragments as needed. Users interested in integrating configuration fragments with the defined semantics only need to define the input models.

Note that the integration semantics is not defined initially in the system configuration profile. Usually, the configuration designer has to define them manually. Reusing the transformation rules for the generation of the integration constraints, as shown in Fig. 8, is another advantage of our work. This automated constraint generation reduces the risk of miss-interpretation by different configuration designers of the integration relations.

The fact that we use the Final Transformation instead of the WModel2Final HOT for constraint generation implies that this constraint generation technique can be used for ATL transformations in general and is not restricted only to weaving and integration transformations.

6 Related work

The idea of data mapping and data integration has been widely investigated in the literature (Omelayenko 2002; Maedche et al. 2002; Spaccapietra and Parent 1994). Defining the mapping between models and model integration can be seen as the successor of the data mapping research. A number of approaches have defined model management operations (such as merging, subtracting, integration, etc.) focusing on the mapping definition between the models and proposing the operations for manipulating the model mappings and the models for different scenarios.

In Rondo (Melnik et al. 2003) the model management operators, such as merge, match, extract, are defined for solving the mapping problem of metadata in XML schemata format. However the defined operators can only create mappings with respect to fixed semantics and they are not flexible enough to represent domain specific mappings. A set of generic model management operations are introduced in (Bernstein 2003) and the author explains how these operations can be applied to the models and their mappings for different application scenarios. The operations are defined in algebra, while the implementation and execution of the abstract operators are left to the users.

A more specific study on defining model management operations for integrating heterogeneous models is discussed in (Reiter et al. 2005). The authors introduce weaving and sewing processes and a set of operators for each process. Their input models are the aspect models and the relations between the models are the cross-cutting concepts of the models. Their weaving process is defined by specific operations (i.e. overrides, references, and prune) between entities and using constraints as pre/post conditions for selecting the entities. Their sewing process connects models using mediators (defined by synchronize or depend operators) without affecting the model entities. However, their weaving concept is different from what we described in this paper and their weaving operators for connecting the model entities are restricted while integrating configuration models may require broader range of connections and links between entities. The integration approach should be extensible to allow the definition of different types of connections with respect to the required properties of the system. We used and extended the weaving concept introduced in (Del Fabro et al. 2005a) which allows for the definition of the extensible corresponding entities that can be translated and executed with model transformations.

In (Jossic et al. 2007; Del Fabro and Valduriez 2007) model weaving is used for integrating software architecture models. The mapping links between the entities of the models are created and then filtered based on some similarities, such as type or name, between the linked entities. Basically, the links are used to map similar entities. In our work however the links between the entities represent the semantics of the relations between the entities and they are more complex and carry target system properties (such as HW availability and/or affinity in the case of AMF and PLM configuration models).

We also capture this semantics as constraints in the integration process. These constraints are used for checking the consistency of the target model when changes are made at runtime.

7 Conclusion

As systems can be developed by the integration of different aspects, services, or components developed separately, the system configuration can be obtained from the integration of the configurations of these different aspects or artifacts developed separately as well. Although developed independently these configuration fragments are interrelated as they represent and potentially act on the same system entities. Moreover, the integration may target specific properties, such as availability or affinity, etc. Therefore, the configuration fragments need to be integrated carefully to form together a consistent system configuration with respect to fragment internal properties and targeted system configuration properties.

We tackled this problem with a model driven approach using model weaving. We used configuration samples from the SA Forum middleware for illustration purposes. Our approach to integrate configuration fragments takes into account the properties of the target system configuration. We used the weaving model to capture the mapping between the elements of the different configuration profiles. We introduced new link types to capture the special relations, i.e. integration semantics, between the entities of these profiles in a weaving model, i.e. they are added to the weaving metamodel. Using a set of ATL transformations we generate a consistent system configuration from the weaving. Although this integration semantics is taken care of in the process of generating the system configuration model, it is not reflected in the system configuration profile. This integration semantics needs to be defined as integration constraints in the system configuration profile in order to guard the consistency of system configuration models against unsafe runtime modifications. This is achieved automatically in our approach.

Our approach for integrating configuration fragments allows for the reuse and the extension of the system configuration generation process as the link types were defined once and reused for the mapping of different entities of the configuration fragments. In the future other profiles can also be added to the process using the same or new link types. The automated generation of system configurations from different input configurations is another advantage, which results in saving time and efforts needed for the task.

The work reported in this paper is part of a larger project aiming at the integration of configuration fragments into a consistent system configuration, validating efficiently runtime modifications to preserve this consistency (Jahanbanifar et al. 2015a), and adjusting automatically when this consistency is violated by the proposed modifications (Jahanbanifar et al. 2016).

Acknowledgment This work has been partially supported by Natural Sciences and Engineering Research Council of Canada (NSERC) and Ericsson.

References

Bernstein, P.A. (2003). Applying model management to classical meta data problems. In: *The conference on innovative data systems research (CIDR)*, Asilomar, California.

Del Fabro, M.D., & Valduriez, P. (2007). Semi-Automatic Model Integration using Matching Transformations and Weaving Models. In: ACM SAC, (pp. 963–970).

Del Fabro, M. D., & Valduriez, P. (2009). Towards the efficient development of model transformation using model weaving and matching transformations. *Software and System Modeling (SOSYM), 8*, 305–324.

Del Fabro, M.D., Bézivin, J., Jouault, F., & Valduriez, P. (2005a). Applying Generic Model Management to Data Mapping. In: *Journées Bases de Données Avancés (BDA)*.

Del Fabro, M. D., Bézivin, J., Jouault, F., Breton, E., & Gueltas G. (2005b). AMW: A Generic Model Weaver. In: *The 1ère Journée sur l'Ingénierie Dirigée par les Modèles*.

Del Fabro, M.D., Bézivin, J., & Valduriez, P. (2006). Weaving models with the eclipse AMW plugin. In: *Eclipse Modeling Symposium, Eclipse Summit Europe*.

DMTF (2013). Standard, open virtualization format spec. Ver: 2.1.0. http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.1.0.pdf.

Eclipse Atlas Model Weaver (AMW). (2010). http://www.eclipse.org/gmt/amw/.

Eclipse Modeling Framework, EMF. (2014). http://www.eclipse.org/emf.

Jahanbanifar, A., Khendek, F., & Toeroe, M. (2014). Providing Hardware Redundancy for Highly Available Services in Virtualized Environments. In: *8th IEEE International conference on Software Security and Reliability*, Tokyo (SERE), (pp. 40–47).

Jahanbanifar, A., Khendek, F., and Toeroe, M. (2015a). Partial Validation of Configuration at Runtime, In: *18th Int. Symposium on Real-Time Distributed Computing*, Auckland (pp. 288–291).

Jahanbanifar, A., Khendek, F., & Toeroe, M. (2015b). A Model-based Approach for the Integration of Configuration Fragments, Proceedings of the European Conference on Modeling Foundations and Applications (ECMFA), LNCS Vol. 9153, Springer, (pp. 125–136).

Jahanbanifar, A., Khendek, F., & Toeroe, M. (2016). Runtime configuration models adjustment for consistency preservation, in the IEEE HASE'2016 proceedings, Florida, January 07-09, 2016.

Jossic, A., Del Fabro, M.D., Lerat, J.P., Bezivin, J., & Jouault, F. (2007). Model Integration with Model Weaving: A Case Study in System Architecture. In: *The International Conference on Systems Engineering and Modeling (ICSEM)*, IEEE CS Press, (pp. 79–84).

Jouault, F., & Bézivin, J. (2006). KM3: A DSL for Metamodel Specification. In: Formal Methods for Open Object-Based Distributed Systems (FMOODS 2006). Bologna, Italy LNCS, vol. 4037.

Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming, 72*(1–2), 31–39.

Maedche, A., Motik, B., Silva, N., & Volz, R..(2002). MAFRA—AMApping FRAmework for Distributed Ontologies. In: *13th International Conference on Knowledge Engineering and Knowledge Management*, Sigüenza (pp. 235–50).

Melnik, S., Rahm, E., & Bernstein, P. (2003). Rondo: A Programming Platform for Generic Model Management. In: *SIGMOD conference*, (pp. 193–204).

Nechypurenko, A. et al. (2007). Application of aspect-based modeling and weaving for complexity reduction in development of automotive distributed real-time embedded system. In: *The proceedings of AOSD'2007*, Vancouver.

Object Management Group. (2011). Unified Modeling Language – Superstructure Version 2.4.1, formal/2011–08-05, http://www.omg.org/spec/UML/2.4.1/.

Omelayenko, B. (2002). A Mapping Meta-Ontology for Business Integration. In: *The workshop on knowledge transformation for the semantic Web (KTSW 2002) at the 15th European conference on artificial intelligence*, Lyon, (pp. 76–83).

Reiter, T., Kapsammer, E., Retschitzegger, W., & Wimmer, M. (2005). Model integration through mega operations. In: *Workshop on MDWE 2005*.

Salehi, P., Hamou-Lhadj, A., Colombo, P., Khendek, F., & Toeroe, M. (2010). A UML-Based Domain Specific Modeling Language for the Availability Management Framework. In: *12th IEEE International High Assurance Systems Engineering Symposium*, (pp. 35–44).

Service Availability Forum. (2010a). http://www.saforum.org.

Service Availability Forum. (2010b) Availability Management Framework, http://www.saforum.org/HOA/assn16627/images/SAI-AIS-AMF-B.04.01.pdf.

Service Availability Forum (2010c) Platform Management Service, http://www.saforum.org/HOA/assn16627/images/SAI-AIS-PLM-A.01.02.pdf.

Spaccapietra, S., & Parent, C. (1994). View integration: a step forward in solving structural conflicts. In: *IEEE Transactions on Data and Knowledge Engineering*, *6*(2): 258–274.

Tisi, M., Jouault, F., Fraternali, P., Ceri, S., & Bézivin, J. (2009). On the use of higher-order model transformations. In R. F. Paige, A. Hartman, & A. Rensink (Eds.), *ECMDA-FA 2009. LNCS* (Vol. vol. 5562). Berlin: Springer.

Toeroe, M., & Tam, F. (2012). *Service availability: principles and practice*. New York: Wiley and Sons.

Azadeh Jahanbanifar obtained her PhD in computer science from Concordia University, Canada and MSc in information technology from Amirkabir University of Technology, Iran. Her research interests include model driven engi-neering, model integration, validation and adaptation. Azadeh has several years of work experience in software development as an analyst and designer of electronic banking services and high level service management systems.

Ferhat Khendek received his PhD from University of Montreal, Canada. He is a full professor in the department of Electrical and Computer

Engineering of Concordia University where he also holds since 2011 the NSERC/Ericsson Senior Industrial Research Chair in Model Based Management, a major collaboration between Ericsson and Concordia University. Ferhat Khendek's research interests are in model based software engineering and management, formal methods, validation and testing, cloud computing, real-time software systems, and service engineering and architectures.

Maria Toeroe is an Expert at Ericsson working in the area of dependable software, service availability and fault tolerance. She has represented Ericsson in the Service Availability Forum and more recently in OPNFV. Maria is also the technical coordinator of the Ericsson research collaboration with Concordia University. She has numerous publications and served as organizer and program committee member of different conferences. Maria holds a PhD from the Budapest University of Technology and Economics.