Enhancing Architecture Design Decisions Evolution with Group Decision Making Principles

Ivano Malavolta[1], Henry Muccini[2], Smrithi Rekha V. [3]

[1]Gran Sasso Science Institute, L'Aquila, Italy
[2]Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy
[3]Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, India
ivano.malavolta@gssi.infn.it, henry.muccini@univaq.it, smrithirekha@gmail.com

Abstract. In order to build resilient systems, robust architectures are needed. The software architecture community clearly recognizes that robust architectures come from a robust decision-making process. The community also acknowledges that software architecture decision-making is not an individual activity but a group process where architectural design decisions are made by groups of heterogeneous and dispersed stakeholders. The decision-making process is not just data driven, but also *people driven*, and group decision making methodologies have been studied from multiple perspectives (e.g., psychology, organizational behavior, economics) with the clear understanding that a poor-quality decision making process is more likely than a high-quality process to lead to undesirable outcomes (including disastrous fiascoes).
In this work, we do propose to explicitly include group decision making strategies into an architecting phase, so to clearly document not only the architectural decisions that may lead to the success or failure of a system, but also group decision making factors driving the way architecture design decisions are made. In this respect, this work defines a group design decision metamodel (for representing group design decisions and their relationships), together with ways to trace group design decisions towards other system life-cycle artifacts, and a change impact analysis engine for supporting evolving design decisions.

1 Introduction

Dependability and resilience in software engineering have been analyzed since a long time from a (purely) technical perspective, by proposing architectures and processes for realizing dependable[1] and resilient systems[1, 2], by modeling and analysing properties of resilience [3, 4], by monitoring the system state at run-time [5], and so on.

More recently, human and social aspects are being considered as an important factor when developing quality systems. The role of the human beings in automated software testing has been the topic of a Dagstuhl seminar [6]. The role of socio-technical coordination has been remarked by James Herbsleb in his keynote at ICSE 2014 (the 36th International Conference on Software Engineering) and through his publications. The

[1] Architecting Dependable Systems series of workshop: http://www.cs.kent.ac.uk/people/staff/rdl/ADSFuture/resources.htm

Social Software Engineering workshop is at its sixth edition, being this year co-located with the Foundations of Software Engineering (FSE 2014) conference. The way people work together impacts their productivity and the quality of the outcome: group decision making processes and methods have been carefully analyzed in a number of areas [7, 8].

Along these lines, we have been recently analyzing *group decision making* principles in the *architecture design decision* process. Architecture design decisions (ADDs) are considered first class entities when architecting software systems [9–11]. They do capture potential alternative solutions, and the rationale for deciding among competing solutions. It has been recognised that ADDs have to be explicitly documented and kept synchronized with (potentially evolving) requirements and the selected architectural solution. Group decision making (GDM), instead, consists of a set of methods and principles driving the way groups make collaborative decisions. In recent work, we have been analyzing how group decision making principles and methods have been implemented in state-of-the-art ADD approaches. More specifically, in [12] we have been interviewing a number of architects working in (or collaborating with) industry to identify how architectural decisions are taken. As expected, most of the architectural decisions are made in groups, distributed or co-located, decisions involving a number of different stakeholders.

This work, by building upon our previous work presented at SERENE 2011 [13] (where an approach to support ADD evolution has been discussed), proposes an approach to incorporate GDM strategies explicitly into evolving architecture design decisions. It proposes a reference metamodel to group decision making, that describes the minimal reasoning elements necessary to suitably realize a group decision making approach as part of an architecture design decision process. Then, it proposes an approach to create traceability links between architecture design decisions and other artefacts as a means to support the propagation of changes from one artifact to another. Starting from the known fact that systems to be resilient require robust architectures, we do here argue that a well defined, explicit, and systematic architecture design decision process, with an explicit knowledge on group decision making strategies, and the ability to evolve, may help to design a longer living, more resilient, architectures.

The contribution of this work is mainly in the following lines: a) it provides a metamodel to include GDM principles explicitly into architecture design decisions (therefore, enabling to extend current ADD methods to take into consideration GDM principles and methods); b) it describes how the relationships among ADDs can be explicitly represented using the defined metamodel, and c) it defines bidirectional traceability links between ADDs, requirements and architectural elements which will help in analyzing the impact of evolution on those artifacts.

This work extends our SERENE 2011 contribution [13] by explicitly taking into account the group decision making principles we had highlighted in our recent papers [14, 12]. This implies that the ADD metamodel presented in [13] had to be extensively extended in order to include a number of missing GDM factors. The traceability and change propagation engines had to be highly revised in order to incorporate the new needs dictated by GDM. We expect this work to provide the the baseline to build new approaches and tools for group-based architecture design decisions.

The main components of this paper are five sections, organized as follow. Section 2 provides background information on architecture design decisions and group decision making. Section 3 presents the main contribution of this work, in terms of a metamodel for group decision making in software architecture. Section 4 introduces some usage examples to show how this proposal can be used in practical terms. Related work are presented in Section 5, while conclusions and future work are discussed in Section 6.

2 Background

The SA research community is currently looking at SA as a decision-making process involving a group of stakeholders [12]. This needs a revision of the metamodels to accommodate group decisions which will in turn impact the SA decision methods and tools. Group Decision Making enjoys a significant position in management research. GDM research has been focusing on enhancing group characteristics, contribution of group members, GDM process and thereby the outcome. Research points that decisions made by groups are often perceived to be more reliable than those by individuals. Groups are considered to be means of aggregating individual view points as well as combining multiple perspectives to make high quality decisions [15]. Groups are better than individuals because:

- Groups have access to bigger pool of information
- Groups bring in diverse perspectives
- More alternatives may be uncovered
- More reliable as multiple people are involved before a decision is made

Researchers also recognize that there may be some issues with GDM. It may take more time to arrive at consensus when more people are involved and conflicts may arise when multiple perspectives are presented [8]. Hence focusing on the GDM process helps to resolve these issues. Several well tested GDM methods exist which are used by organizations. GDM methods are broadly categorized as semi-formal methods like brainstorming, ranking based methods like voting or nominal group technique and structured methods like Analytics Hierarchy Processing [7]. There are numerous GDM methods and we discuss a few here.

- Brainstorming: This is a semi-formal method were participants discuss the issues and brainstorm solutions. the discussions are moderated by a leader.
- Voting: Participants indicate their preferences on a pre-determined set of alternatives through votes.
- Nominal Group Technique (NGT): This technique involves allotting time to individual participants to think and present their viewpoints (preempting the others from speaking) and then discussing as a group.
- Delphi Technique: This is an iterative process involving several rounds of discussions with each round involving feedback from previous iterations.
- Analytic Hierarchy Processing (AHP): This is a structured technique that involves pairwise comparison of alternatives and criteria and weighing the alternatives based on the criteria before making a decision.

Given this important role played by GDM methods in enhancing the outcome of the decision-making process, it is important the SA community recognizes the need to integrate GDM into current SA decision methods. This is especially valuable in the context of rapidly evolving software. Software evolution also includes the evolution of the underlying architecture, the decisions and the rationale. When multiple stakeholders are involved, it may be important to factor in their preferences before making changes to the decisions and thereby the resulting architecture. Hence we need to explicitly model multiple stakeholder viewpoints in the SA decision-making process. Current metamodels, though they recognize the involvement of multiple stakeholders, do not have explicit integration of GDM. This calls for a modification of the existing metamodels of SA decision-making to include various aspects of GDM.

3 Architectural evolution and Group Decision Making

Given a generic architecture design decision approach, a group decision making (GDM) process and method builds on top of it to regulate how multiple stakeholders make decisions when working together. In this light, the proposed extension (to our SERENE 2011 work in [13]) aims at the straightforward integration of existing group decision making (GDM) methods into architecture design decision approaches.

Figure 1 gives an overview of the approach for architectural group design decisions evolution.

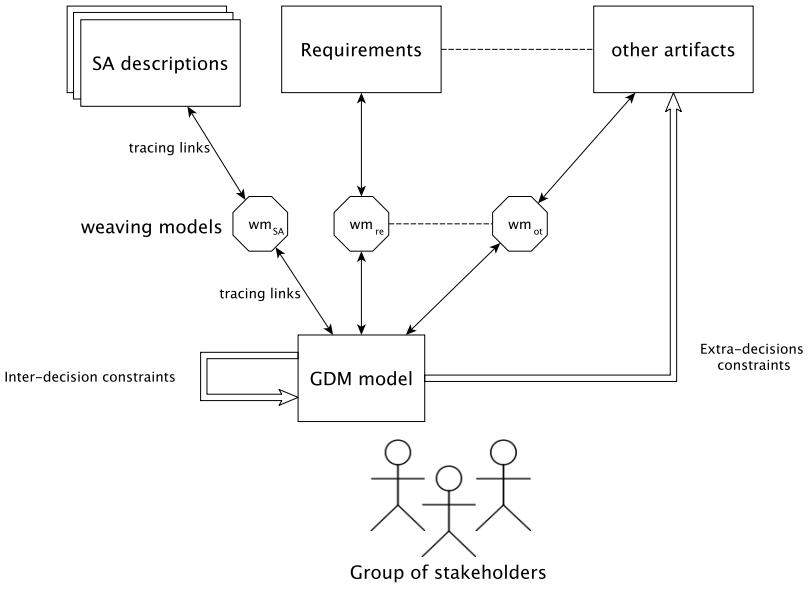

Fig. 1. Overview of the approach

The approach foresees three main components: the *GDM model*, *traceability links* between software artifacts, and *a change impact analysis* approach.

The **GDM model** includes the GDM principles, and all the design decisions resulting from the collaboration of groups of stakeholders during the design process. The GDM model also classifies design decisions depending on their type and their status, and keeps track of the various relationships between them (Section 3.1 gives an overview on how we extended the design decisions model). Design decisions can be linked to any kind of modeling artifact produced throughout the system development life-cycle; for example, design decisions can be linked to the specification of the requirements of the system, to a set of SA descriptions resulting from the chosen design decisions (only one SA description is chosen, all the others are alternatives), etc.. In this specific work we primarily focus on group design decisions and their evolution, so it basically treats other modeling artifacts as black boxes. The only requirement posed by the proposed approach is that each artifact that has to be considered during the evolution analysys must be represented as a distinct model. This requirement is in line with the everything-is-a-model principle of Model-Driven Engineering [16].

Traceability is realized as a set of tracing links from each modeling artifact to the GDM model (see the arrows in Figure 1). *Tracing links* are contained into special models (e.g., wm_{SA}, wm_{SA}, and wm_{ot} in Figure 1) called *weaving models*. A unique aspect of our approach is that tracing links are represented as models; also this aspect is perfectly in line with the everything-is-a-model MDE principle [16], and represents an interesting added value for our approach.

In particular, the use of weaving models allows designers to keep the models of the artifacts *free from any traceability metadata*. This makes the proposed approach *independent from the used modeling languages*, allowing stakeholders to use their preferred notation to describe the system according to their specific concerns. Moreover, the use of weaving models opens up for an **accurate evolution impact analysis** since they provide traceability information both at the model-level and at the model-element level. Intuitively, a designer not only can specify that a design decision dd_x pertains to a whole architectural model am, but also that dd_x pertains only to components c_1 and c_2 in am. In this paper we will not go into the details of this aspect of the approach because it is out of the scope of the paper. The interested reader can refer to [13] for further details on how weaving models are an added value of the proposed approach.

Arranging architecture-related artifacts using models and weaving models enables us to reformulate the design decisions evolution[2] impact analysis into the activities of: identifying and checking which design decisions and which parts of other architecture-related artifacts are not aligned with the evolved GDM model. Changes in an GDM model element can have an impact either on elements (e.g., design decisions conflicting with a deleted decision become valid alternatives), or also to external related artifacts

[2] In this work we consider *evolution* to be a set of incremental changes of a model in response to changing external factors [17], as it is considered in the model management research area; these external factors can include changes to requirements, issues, technology availability, and so on.

(e.g., a component realizing a rejected decision must be checked and probably modified).

In the next sections we describe the GDM-enhanced design decision metamodel (Section 3.1), our proposal to trace design decisions towards other system life-cycle artifacts (Section 3.1), our extension of the change impact analysis engine for supporting GDM (Section 3.1) and the high-level software architecture of the tool realizing the whole approach (Section 3.2).

3.1 Extended Metamodel to Support GDM

In MDE a model always conforms to a metamodel, so we propose the GDM metamodel for representing group design decisions and their relationships. In the proposed approach design decisions are represented within a GDM model. A GDM model represents:

- all the involved groups of stakeholders,
- GDM sessions,
- (categorized) design decisions with their status and relationship with respect to other design decisions,
- stakeholders' preferences with respect to some design decision,
- all the relevant design reasoning history of the system (e.g., status of design decisions, logs, references, etc.).

In line with [13], for the sake of generality, the GDM metamodel is notation- and tool-independent so that it is able to represent design decisions and their evolution in a general manner.

The GDM metamodel has been defined by analyzing the current state of the art in GDM (see [14]) and design decisions representation and reasoning (e.g., [9], [18], [19]). We designed GDM so that it (i) presents the most common aspects and relationships of design decisions, and (ii) contains concepts to define the evolution of an ADD (like its state, history and scope).

Figure 2 graphically shows the metamodel for group design decisions models. As already said, it is an extension of the generic metamodel presented in [13], the coloured metaclasses belong to the extension we made, whereas yellow metaclasses are part of the base metamodel being extended.

For what concerns the **base part of the metamodel**, `DesignDecision` is its main concept. It contains attributes like *description*, *state* that represents the current state of the decision (e.g., idea, tentative, decided, rejected, etc.), *timestamp* that specifies when the design decision has been created, *history* that keeps track of the evolution history of the decision (mainly, author, timestamp and status of each past version of the decision, and so on). The *evolved* attribute is used as a marker to identify which design decisions have been subject to an evolution. This will be used during the evolution impact analysis, which is in charge of managing traceability between evolutions of architectural design decisions models (see [13] for further details on the GDM model evolution management).

Each design decision can be related to some other design decisions by means of typed relationships, e.g., *constrains*, *enables*, *conflictsWith*, and so on (please refer to

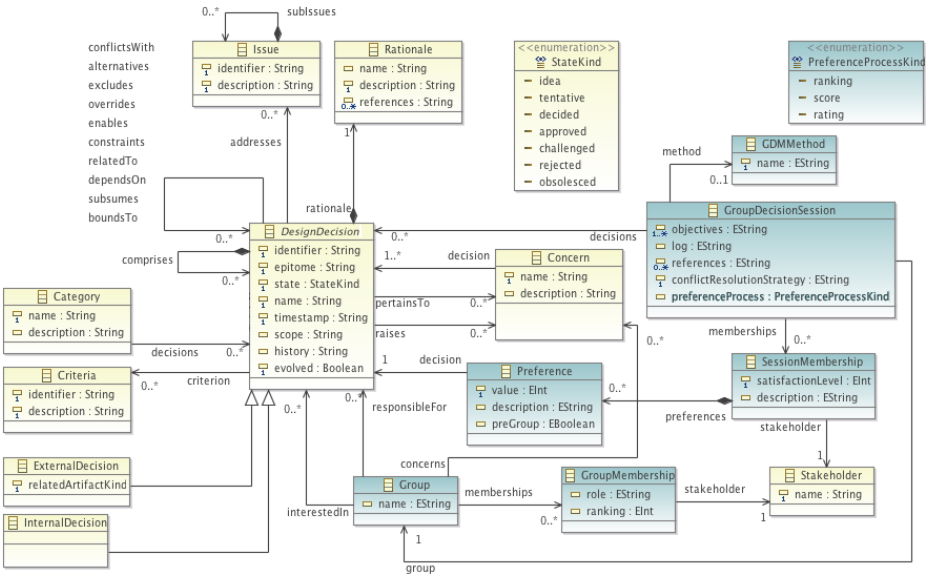

Fig. 2. Generic metamodel for supporting software architecture decision making

[9] for a comprehensive description of all the types of relationships). The *comprises* relationship exists when a high-level decision is composed of a set of more specific decisions; differently from all the other relationships (which are represented as simple reference), it is represented as a composition reference because the lifetime of comprised decisions must correspond to the lifetime of the comprising decision [13].

The GDM metamodel distinguishes between internal and external group design decisions. This distinction is useful during evolution impact analysis since it allows the tool to tune the scope of the analysis depending on which design decisions have been evolved.

Designers must associate a `Rationale` to each design decision in order to keep track of the reasons behind the decision. In order to keep design decisions cognitively manageable and well scoped, they can be organized into `categories`. `Stakeholders` are the member of a group, which will be described later in this section.

The **extension of the metamodel** takes into consideration the aspects related to group decision making in the context of architecture description. In the following we describe the elements of our extension.

A `Group` represents the a collection of stakeholders (e.g., software architects, developers, designers, testers, users, etc.) who have regular contact and mutual influence when working together to achieve a common set of goals. The *memberships* relationship represents the participation of each stakeholder in the group. Each group member can have also a set of *concerns* identifying the key design problems related to or raised

by a specific design decision. A group or group member can be either *responsible for* or *interested in* a set of design decisions.

`GroupMemberships` put in relationship each group with its *stakeholder*; in other words, it links a group with each of its members. As suggested in [14], it would be useful to account for hierarchy or expertise differences among stakeholder, i.e., a *ranking*. Indeed, it would be useful to prioritize stakeholders based on some criteria like seniority or expertise to make the process more robust[7]. However, since some organizations may have flat structures where all stakeholders enjoy equal priority, the ranking attribute is optional. Also, each member of a group can optionally have a *role* within the group she belongs to; this attribute can be used by a reasoning engine so that it can prioritize decision-makers, senior architects, and so on. Each stakeholder can belong to multiple groups.

A `GroupDecisionSession` represents a single continuous meeting, or a series of meetings of a *group* of stakeholders. Each group decision session has a set of *objectives* because in any decision making exercise, every member needs to be clear about the objectives before the start of the process. In addition to the set of established objectives, the main outcome of a group decision session is the set of design *decisions* identified, modified or, more in general, related to it. Also, each group decision session has an history *log* and a set of *references* in order to allow session members to carefully record all the activities performed.

Conflicts are inherent to GDM: appropriate conflict resolution strategy could be applied to the used SA decision-making method [14]. In this context, the *conflictResolutionStrategy* attribute uniquely identifies the conflict resolution strategy applied in the group decision session. The most popular strategy is the collaborative style of conflict resolution [12].

Stakeholders participating in GDM shall be enabled to indicate preferences [14]. For example, in [20] the alternative scenarios are scored and the stakeholders vote the alternatives and the voting method has been chosen to enable the groups to arrive at consensus in a timely manner. According to this, each group decision session has a *preferenceProcess* attribute describing the process used for recording the preferences of each stakeholder. Preference processes can be based either on a *ranking*, *scoring*, or a *rating* system.

The `GDMMethod` is referenced by a group decision session and it represents the specific architecture design decision method used for evaluating the various design decisions considered during the session. Some popular GDM methods are: brainstorming, voting, Nominal Group Technique (NGT), Delphi technique, Analytic Hierarchy Processing (AHP), etc. GDM methods can be shared across different group decision sessions within the whole organization/project.

Each stakeholder participating to a group decision session is linked to the session itself via a `SessionMembership` element. This intermediate element is keeps track of the *satisfaction level* that the stakeholder achieved during the session. Indeed, literature points that satisfaction of group members is very important [14], so something to indicate satisfaction levels with the outcome is very important. Moreover, a *description* of the experience of the stakeholder while participating to the group decision session is recorded, together with an indication of their *preferences*.

A `Preference` is composed of its cardinal value (so that it can be easily compared with respect to the preferences of other stakeholders within the group) and its description. Also, stakeholders often come with certain preferential biases before the GDM process [21] and during the discussion as more and more information is exchanged, stakeholders tend to revisit their preferences; along this line, the *preGroup* boolean attribute is *true* when the preference has been expressed before the group decision session.

Providing a concrete syntax for GDM models is out of the scope of this work, so we build on and reuse the simplified graph-based representation presented in [13].

Tracing group design decisions to other modeling artifacts. The need for tracing design decisions to/from other system life-cycle artifacts is well understood, especially when those artifacts are requirements specifications or architecture descriptions. As introduced in the beginning of this section, in our approach tracing links are contained into special models (technically called weaving models). A weaving model can be seen as a means for setting fine-grained relationships between models and executing operations on them. The types of relations that can be expressed in our weaving models have been described in details in [13]. Basically, each weaving model has a reference to the GDM model, a reference to the modeling artifact being woven, and a set of links. Each link represents a fine-grained, many-to-many, traceability link between one or more group design decisions and one or more element of the linked modeling artifact. For example, a weaving model may allows designers to associate many architectural components to a single design decision, to assign many design decisions to a single component, or to assign a single design decision to a single component. Each link can be either a *tracing* link if the linked artifact elements trace to/from the linked design decisions, or a *conflict* link if the linked artifact elements are not compliant with the linked design decisions [13].

The use of weaving models to store links between decisions and other artifacts allows us to keep design decisions sharply separated from other artifacts. Furthermore, designers now can focus on GDM models when they need to reason on ADD-specific issues, and on other modeling artifacts when they wants to focus on other aspects of the system under construction. We believe that this distinction helps in keeping all the artifacts well organized and in simplifying the effort to perform the various architecting activities.

Group decisions impact analysis. In our approach design decisions validation is performed by executing a set of constraints ruling on the involved elements to be checked [13]. Since the proposed approach heavily relies on MDE, and since we want designers to specify constraints with an high-level language, these constraints are defined in the Object Constraint Language (OCL[3]), an OMG standard language used to describe expressions on models.

In this specific context, we can distinguish between **Inter-decisions constraints**, and **Extra-decisions constraints**: the first kind of constraints is executed on GDM

[3] Object Constraint Language specification: `http://www.omg.org/spec/OCL`. Verified in July 2014.

models and aims at checking the internal consistency of GDM models, whereas the second one is executed on weaving models and aims at checking and identifying which elements in other modeling artifacts are impacted by the evolved GDM model.

```
1 context GroupDecisionSession
2   inv allMembersPreferenceDefined :
3     not self.memberships->exists(e | e = e.preferences->notEmpty
        ())
```

Listing 1.1. Example of OCL contraint validating Design Decisions

In the extended version of the approach, we added a set of OCL constraints with the aim to check the extended GDM model with respect to both inter-decisions and extra-decisions consistency. For example, the listing above shows an inter-decision constraint that checks every instance of GroupDecisionSession within the GDM model, and checks if all its members have expressed at least a preference with respect to a design decision.

The results of this validation give stakeholders insights and accurate information about which element (being it an architectural element, a requirement, or any other modeling artifact) is not aligned with the evolved design decisions. It is up to the designer now to analyze the information provided by the analysis step and to start a reasoning process with the goal of reaching a certain level of stability among the involved artifacts and the evolved GDM model. Within this process, the OCL-based validation engine helps stakeholders to be reasonably confident about the validity of the system being architected even when design decisions are evolving because any conflicting decision is immediately flagged.

Finally, it is important to note that the set of constraints is an *open set*: designers may add their own constraints depending on the specific project they are working on, or also depending on company-specific internal regulations. the validation engine will automatically consider newly added constraints. This choice allows designers to reuse their knowledge about OCL, without forcing them neither to use languages at a lower level of abstraction nor to learn a new ad-hoc language to specify DD validation procedures.

3.2 Tool support

Figure 3 shows an overview of the main components of the prototype tool we are implementing for supporting the approach presented in this paper. More specifically, the current version of the prototype realizing has been implemented as a set of Eclipse[4] plugins. Eclipse is an open-source development platform comprised of extensible frameworks and tools. We chose Eclipse as base platform as (i) it is based on a plug-in architecture, thus enabling us to easily extend it for our purposes and to distribute our tool in a straightforward manner, and (ii) it allows us to reuse many off-the-shelf components for our purposes, especially when considering MDE engines. Indeed, metamodels can be defined using the Eclipse modelling Framework (EMF)[5], a Java framework and code

[4] `http://eclipse.org`. Verified in July 2014.

[5] `http://www.eclipse.org/modeling/emf`. Verified in July 2014.

generation facility for MDE-based tools. Moreover, many architectural languages support the EMF modelling framework [22]. Other reused off-the-shelf components will be described later.

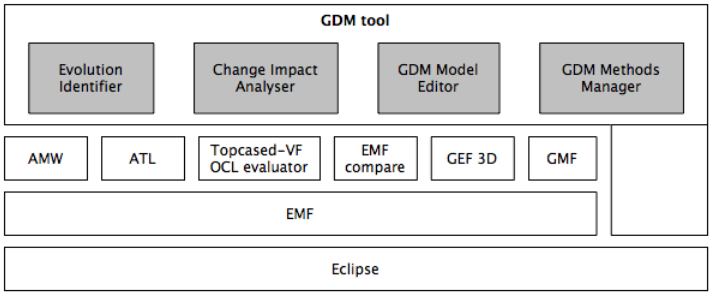

Fig. 3. Tool support

The GDM tool prototype is composed of four main components, each of them devoted to manage a specific aspect of our approach. More specifically, the **Evolution Identifier** component has the goal of identifying the portions of GDM model that are involved in an evolution round, and to make them distinguishable by the other components within the tool. The *Evolution Identifier* component is implemented as a combination of model-to-model transformations, a model comparison technology. The identification of the evolved GDM elements is a two-steps process: comparison of the GDM model with its latest stable version and evolved design decisions identification. The comparison stap is based on EMF Compare[6], an Eclipse plug-in including a generic comparison engine and a facility for creating the corresponding delta model; this delta may itself be represented as a model. As described in [13], evolved design decisions within a GDM model are identified by their *evolved* attribute set to *true*. So, we implemented a model-to-model transformation that takes as input the GDM model and the delta model, selects those design decisions that are related to relevant evolved GDM elements, and updates the GDM model by setting to *true* the *evolved* attribute of all evolved decisions. This transformation is specified using the Atlas Transformation Language (ATL) [23], an hybrid model transformation language.

The **Change Impact Analyser** component is in charge of extending the set of evolved design decisions in the GDM model with respect to all the modeling artifacts used throughout the system development life-cycle. Also this component is based on the ATL model transformation language The identification of the element impacted by a design decision evolution is performed by means of a set of transformations similar to the one realizing the *Evolution Manager* component. The only difference is that in this case the model transformation marks weaving links, instead of GDM model elements.

[6] `http://www.eclipse.org/emf/compare`. Verified in July 2014.

As explained in Section 3.1, weaving links are stored in weaving model. The technology we use for managing weaving models is the Atlas Model Weaver (AMW) [23]. The OCL constraints execution is based on the Topcased-VF OCL evaluator[7].

The **GDM Model Editor** component exposes exposes all the modeling capabilities to stakeholders, so that they can graphically define GDM models in a straighforward manner. This component is based on the well-known Eclipse Graphical Modeling Framework (GMF[8]), a dedicated framework providing generative components and runtime infrastructures for developing graphical editors based on EMF. The GDM model editor presents a graph-based graphical editor to stakeholders, and provides facilities like elements creation, management of their properties, copy-and-paste, export to an image file, etc.

Finally, the **GDM Methods Manager** component aims at providing a certain degree of automation to the GDM process. More specifically, it allows stakholders to automatically compute the rankings of all the design decisions involved in a group decision session, depending on the specified GDM method. So, for example if the group decision session has AHP as GDM method, then the GDM Methods Manager executes the pairwise comparison and ranking of alternatives, and mathematically computes the best solution at run-time. Clearly, this facility helps in both boosting the group decision sessions, and in always having a correct design decision ranking. This component is implemented as a set of dedicated Java classes, each of them implementing a specific GDM method, and a set of generic Java classes for updating the GDM model at run-time, i.e., in parallel with stakeholders using the GDM model editor.

4 Benefits of GDM for resilience

In this section we discuss how including GDM in SA decision-making metamodel is beneficial to manage evolving systems. As mentioned in earlier sections, while managing the evolution of sofware systems, we need to manage the evolution of the architecture together with the design decisions and design rationale. Not only that, since these decisions are made by a group of people, all their viewpoints need to be considered before making changes. In the state-of-the-art metamodels, since GDM is not visibily modeled, changes to decisions is assumed to be a straightforward process which concerns only modification of either the requirements or decisions or the architectural elements. Hence SA knowledge was captured only about these aspects. This has a direct bearing on the tools created based on the metamodels and hence very few of them support GDM. Integrating GDM into our previous SA evolution metamodel may be beneficial in several scenarios some of which are:

- In a multi-stakeholder system, each stakeholder proposes different set of alternatives, criteria and preferences about solutions. When changes are proposed to a system, not only do the requirements and artefacts evolve, but also the decisions made by these group members. When these decisions are modified or discarded, it may need a revisiting of the alternatives proposed by all stakeholders before choosing the next best alternative. Hence explicit modeling of GDM simplifies this process.

[7] `http://gforge.enseeiht.fr/projects/topcased-vf`. Verified in July 2014.
[8] `http://www.eclipse.org/modeling/gmp`. Verified in July 2014.

- When decisions or criteria need modifications, it may not only result in conflicting decisions but may demand modifications in preferences indicated by stakeholders. This may need an iterative GDM process which allows the revisiting of preferences to support evolution.
- Software Architects prefer making timely decisions. When the architecture evolves, it may be necessary to make time-bound decisions to quickly accommodate the changes. When tools are built based on the extended metamodel, they would assist quick decision-making using one of the listed GDM methods.
- Evolution is often an expensive process. When multiple stakeholders are involved, they have more information to decide from and hence may suggest modifications to only those aspects of the system that are absolutely necessary thereby moderating the cost of evolution.
- Fundamentally GDM assumes diverse objectives. While architectures evolve, they may have a direct impact on these varied objectives of multiple stakeholders. Hence a robust metamodel with GDM may be required to balance conflicting objectives.

5 Related Work

In our previous work on GDM [20], we had presented our study of 29 practitioners and researchers involved with architecting of Software Systems. The study was mainly to understand the state-of-the-art GDM practices in the SA community, how far it matches with methods discussed in literature and what challenges the community faces. This study has greatly helped us in understanding actual needs of the industry. As per our findings, GDM in Software Architecture is a combination of formal and informal methods where a group of stakeholders with diverse expertise make decisions. The community acknowledges that the current tools are not sufficient to assist in GDM and hence valuable knowledge about the process is lost. They also face several challenges and often find themselves entering into conflicts but have limited expertise to enhance their process to handle these issues. We further extended our study in [14] to evaluate current SA decision-making methods discussed in literature to see if they can support GDM. We found that with the exception of a few methods, most of them are not yet suitable to accommodate group decisions.

Tofan et al, in [24], have done an extensive and systematic study of state of research in SA decisions. They have analyzed about 144 papers that discuss SA decisions and categorized them based on six research questions. They found that only 22 of them broadly discuss group decision-making and hence there is lot of scope for improvement in the fundamental SA decision-making methods to include groups.

Several Architectural Knowledge (AK) management tools have been compared in [25]. The authors have used ten criteria to compare the tools. They mention that the current version of most of the tools, which are based on the IEEE 42010 as well as on SA decision-making metamodels, lack proper support for collaborative sharing of knowledge.

6 Conclusions and Future work

Robust architectures come from a robust decision-making process.

Two important factors (among others) impact the decision process robustness: collaborating people shall work following explicitly defined GDM principles and processes in order to reach a consensus; decisions (as well as other artefacts) may evolve over time and need to be kept in synch. In this line, this work has presented an extension to the evolving architecture design decision framework presented in [13]. Such an enhancement has requested to extend the metamodel with new meta-elements and meta-relationships, to extend the traceability and traceability links previously proposed with new weaving models, and to extend and revise the impact analysis approach. A sketch of the tool architecture has been presented as well.

As far as concern future work, we are working towards the design and implementation of a collaborative working environment, where different distributed architects and decision makers can participate to the collaborative architectural design of a software system. Such a framework will enable the architects to document the GDM practices in use, to store alternative and selected decisions, to report on changes and how they may impact other software artifacts. More than this, the framework will include the possibility to create new plugins, each one implementing a specific GDM method.

References

1. Cirani, S., Fedotova, N., Veltri, L.: A resilient architecture for dht-based distributed collaborative environments. In: Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems. SERENE '08, New York, NY, USA, ACM (2008) 1–8
2. Stoicescu, M., Fabre, J.C., Roy, M.: Architecting resilient computing systems: Overall approach and open issues. In Troubitsyna, E., ed.: Software Engineering for Resilient Systems. Volume 6968 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 48–62
3. Rodríguez, R.J., Merseguer, J., Bernardi, S.: Modelling and analysing resilience as a security issue within uml. In: Proceedings of the 2Nd International Workshop on Software Engineering for Resilient Systems. SERENE '10, New York, NY, USA, ACM (2010) 42–51
4. Prokhorova, Y., Troubitsyna, E.: Linking modelling in event-b with safety cases. In: Proceedings of the 4th International Conference on Software Engineering for Resilient Systems. SERENE'12, Berlin, Heidelberg, Springer-Verlag (2012) 47–62
5. Hamida, A.B., Bertolino, A., Calabrò, A., De Angelis, G., Lago, N., Lesbegueries, J.: Monitoring service choreographies from multiple sources. In: Proceedings of the 4th International Conference on Software Engineering for Resilient Systems. SERENE'12, Berlin, Heidelberg, Springer-Verlag (2012) 134–149
6. Harman, M., Muccini, H., Schulte, W., Xie, T.: 10111 executive summary – practical software testing: Tool automation and human factors. In Harman, M., Muccini, H., Schulte, W., Xie, T., eds.: Practical Software Testing : Tool Automation and Human Factors. Number 10111 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2010)
7. Saaty, T.L., Vargas, L.G.: Decision making with the analytic network process. Springer (2006)
8. Aldag, R.J., Fuller, S.R.: Beyond fiasco: A reappraisal of the groupthink phenomenon and a new model of group decision processes. Psychological Bulletin **113**(3) (1993) 533
9. Kruchten, P.: An Ontology of Architectural Design Decisions in Software Intensive Systems. In: 2nd Groningen Workshop Software Variability. (October 2004) 54–61

10. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: WICSA 2005. (2005)
11. Potts, C., Bruns, G.: Recording the reasons for design decisions. In: 10th International Conference on Software Engineering ICSE 1988. (1988) 418–427
12. V., S.R., Muccini, H.: A study on group decision-making in software architecture. In: Software Architecture (WICSA), 2014 IEEE/IFIP Conference on. (April 2014) 185–194
13. Malavolta, I., Muccini, H., Rekha, V.S.: Supporting architectural design decisions evolution through model driven engineering. In: SERENE. (2011) 63–77
14. Muccini, H., Rekha, S.: Suitability of software architecture decision making methods for group decisions. In: Proceedings of the 8th European conference on Software architecture. ECSA'14, Berlin, Heidelberg, Springer-Verlag (2014) To appear
15. Brodbeck, F.C., Kerschreiter, R., Mojzisch, A., Schulz-Hardt, S.: Group decision making under conditions of distributed knowledge: The information asymmetries model. Academy of Management Review **32**(2) (2007) 459–479
16. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer **39**(2) (2006) 25–31
17. Levendovszky, T., Rumpe, B., Schätz, B., Sprinkle, J.: Model evolution and management. In: Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems, Berlin, Heidelberg, Springer-Verlag (2010) 241–270
18. Eklund, U., Arts, T.: A classification of value for software architecture decisions. In: Proceedings of the 4th European conference on Software architecture. ECSA'10, Berlin, Heidelberg, Springer-Verlag (2010) 368–375
19. ISO: Final committee draft of Systems and Software Engineering – Architectural Description (ISO/IECFCD 42010). Working doc.: ISO/IEC JTC 1/SC 7 N 000, IEEE (2009)
20. Moore, M., Kaman, R., Klein, M., Asundi, J.: Quantifying the value of architecture design decisions: lessons from the field. In: Software Engineering, 2003. Proceedings. 25th International Conference on. (May 2003) 557–562
21. Stasser, G., Titus, W.: Pooling of Unshared Information in Group Decision Making: Biased Information Sampling During Discussion. Journal of Personality and Social Psychology **48**(6) (June 1985) 1467–1478
22. Lago, P., Malavolta, I., Muccini, H., Pelliccione, P., Tang, A.: The road ahead for architectural languages. IEEE Software **99**(PrePrints) (2014) 1
23. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Proceedings of the MTP, Workshop at MoDELS 2005. (2006)
24. Tofan, D., Galster, M., Avgeriou, P., Schuitema, W.: Past and future of software architectural decisions a systematic mapping study. Information and Software Technology **56**(8) (2014) 850 – 872
25. Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Ali Babar, M.: A comparative study of architecture knowledge management tools. Journal of Systems and Software **83**(3) (2010) 352–370