Measuring Architectural Adaptability in *i** Models

João Pimentel[1,2], Xavier Franch[2], Jaelson Castro[1]

[1] Universidade Federal de Pernambuco - UFPE, Centro de Informática, Recife, Brazil
{jhcp, jbc}@cin.ufpe.br
[2] Universitat Politècnica de Catalunya, Omega-122, CP: 08034, Barcelona, Spain
franch@essi.upc.edu

Abstract. Developing adaptable systems is still a big challenge in software engineering. Different reference architectures and systematic approaches have been proposed to address this challenge. Several of these approaches are based on goal models, given their suitability to express and reason on alternative behaviors. In this paper we intend to provide a basis for comparing architectures described in goal-based models in regard to their adaptability. This way, different approaches to improve adaptability may be compared based on the resulting architectures. To do so we mapped two adaptability metrics onto *i** models and developed guidelines to define the adaptability of individual elements, based on the extra information provided by *i** models. We applied these metrics in a healthcare system to illustrate the comparison of architectures.

Keywords: Models measurement, software architecture metrics, software adaptability, i-star.

1 Introduction

Goal-oriented modeling is widespread in several areas of the software engineering discipline as a suitable way of defining and analyzing organizational expectations and systems requirements [18][5][15][12]. In particular, the *i** framework [13] has become one of the main foundations of goal-oriented modeling, with a strong community and constantly evolving techniques [19][11].

When an *i** model of the system is to be transformed into a software architecture, there are two usual approaches. The first one is to define mechanisms to translate requirements or goal models (in *i**) into other languages used to represent architectures [25][15][8][23], like UML or Acme. This approach benefits of the high familiarity of current developers with the ADLs, their techniques and the massive tool support. The second approach is a seamless one, on which requirements and architecture are expressed on the same language [17][20][4]. This approach uses the high expressivity of the goal models to provide richer architecture models that may, for instance, provide rationales for architectural decisions. Given the suitability of *i** to represent alternative behaviors, in this work we are considering the second approach.

Adaptability is a key concern in autonomic, self-managing or (self-)adaptive systems. Software adaptability may be defined as the software capability for accepting environmental changes. It allows the definition of more flexible, resilient, robust, recoverable and energy-efficient systems [7]. There are some works on modeling adaptive software with *i** [31][21][2]. In this work we are going to define metrics for measuring adaptability in *i** models. The use of metrics is of essential importance to enable the comparison between different models and techniques. Among the different existing approaches to defining metrics in the *i** framework [34][16][10], we are going to use the *i** Metrics Definition Framework method (*i*MDF$_M$), which drives the process of defining metrics derived from metrics defined on a different starting domain. In this case, the starting domain is architecture modeling.

The defined metrics are based on the adaptability index of individual elements of the architecture. In order to guide the analyst in assessing the value for this index, we identified some cases which are indicators of adaptability on individual elements. We also discuss how the extra information provided by extensions on *i**-like languages may as well guide this assessment.

The remainder of this paper is structured as follows. Section 2 briefly describes the *i*MDF$_M$ method, used throughout this work. Next, Section 3 presents how the *i*MDF$_M$ was used in this work, resulting in the definition of adaptability metrics in *i**. An example of application of these metrics is presented in Section 4. Section 5 describes some heuristics to define the adaptability index of an architectural element – first on conventional *i** models, and then with extended *i** versions. Lastly, Section 6 concludes the paper with some final thoughts and future works.

2 The *i*MDF$_M$ Method

The *i*MDF [34] is a framework aimed at guiding the definition of metrics for *i** models, specially focused on the mapping of already existing metrics. This framework is composed of: an extensible *i** metamodel; general forms of *i** metrics; a catalogue of patterns for producing *i** metrics; and the *i*MDF$_M$ method to guide the definition of *i** metrics.

Fig. 1 depicts the steps and related artifacts of the *i*MDF$_M$ method. The *Domain Analysis* step consists of studying the domain of the original metric and mapping its concepts onto the *i** metamodel. In some cases, this will require a customization of the metamodel. The *Domain Metric Analysis* step is concerned with extending the domain ontology, to include some concepts that did not appear in the domain analysis, and to consolidate the domain metrics suite, to prevent any kind of inconsistency. In the *i** *Metrics Formulation* step the domain metrics suite is mapped onto the *i** metamodel and expressed in OCL. The fourth and last step, *iMDF Update*, concerns about updating the language of patterns and the metrics catalogue based on the usage of the *i*MDF framework.

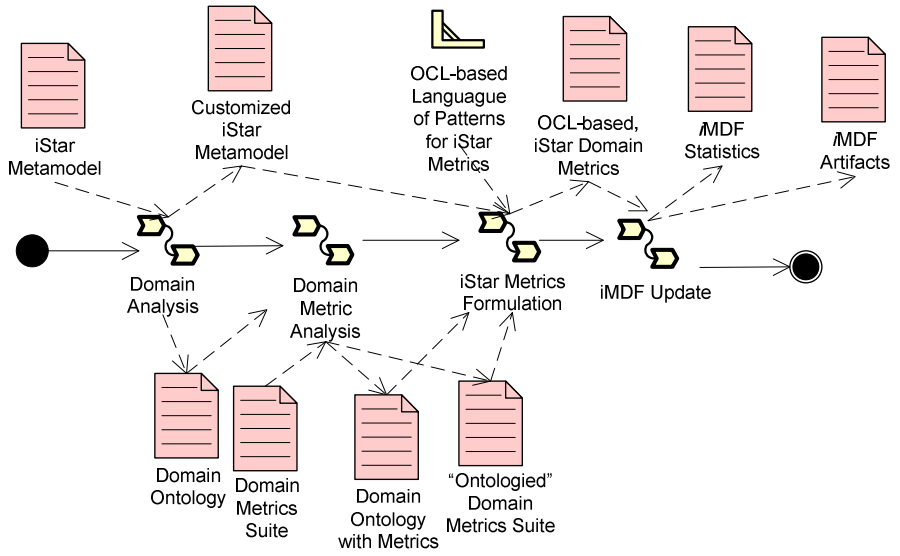

Fig. 1. An overview of the iMDF$_M$ method. It comprises four steps: Domain Analysis, Domain Metric Analysis, *i** Metrics Formulation and iMDF Update. Adapted from [34].

3 Using the *i*MDF$_M$ to Define Adaptability Metrics

In this section we are going to describe the mapping of adaptability metrics from the architecture domain to *i** models using the *i*MDF$_M$ method.

3.1 Step 1 – Domain Analysis

The metrics used in this work are based on conventional architectural models – components and connectors. The mapping between architectures and *i** models is already available in the literature [17][25]. In this mapping, components are represented by *i** actors, and connectors are represented by *i** dependencies. Goal dependencies should be used to represent connectors that describe system-wide features or features pertinent to a large amount of components. If a connector describes or involves processing components it may be represented as a Task dependency. Resource dependencies can represent the flow of data between components. Lastly, softgoal dependencies model connectors that describe or involve properties – such as processing or system properties.

An example of such a mapping is shown in Fig. 2. Each component is mapped onto an actor. The Blood data and Patient data connectors are mapped as resource dependencies, whilst the Alarm connector is mapped as a task dependency.

A software system may have multiple architectural models expressing different concepts, views and abstraction levels, like in the 4+1 view model [30] or in the Conceptual, Module Interconnection, Execution and Code architectures categorization

[9]. Each one of the architectural representations of a system will be mapped into an *i** model. Since in the *i** metamodel [33] the Model is singleton, it was required a change in order to allow the occurrence of multiple models. Fig. 3 gives an overview of this metamodel, highlighting the change on the Model class.

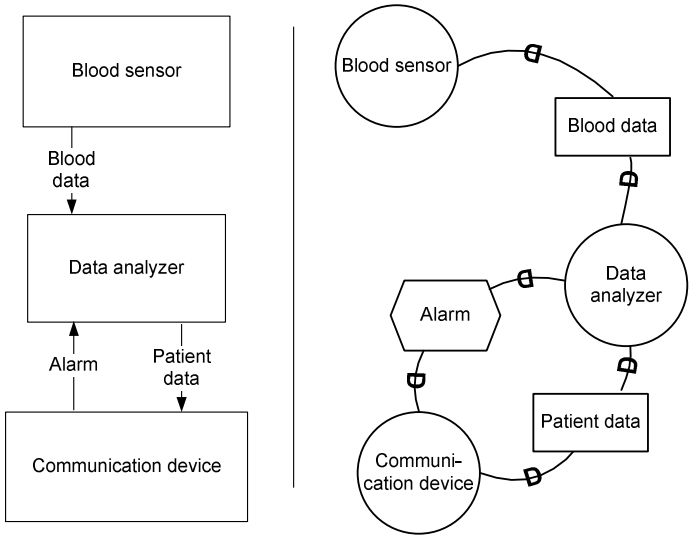

Fig. 2. An example of mapping from an architectural block diagram onto an *i** diagram. The architectural components are mapped onto *i** actors and the architectural connectors are mapped onto *i** dependency links.

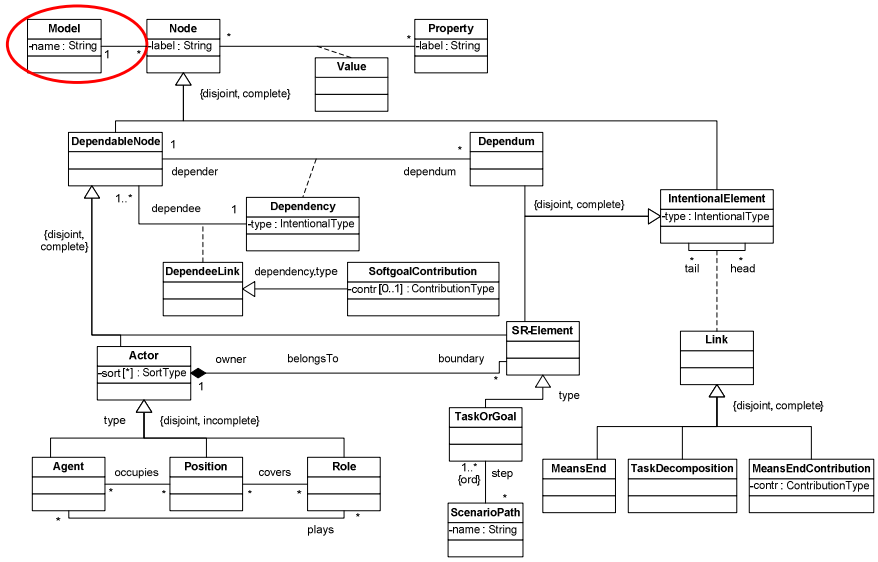

Fig. 3. The *i** metamodel with the change required to allow multiple models. In the original metamodel the Model class was singleton.

Some of the *i** concepts - such as Means-End Links and Task Decompositions - will not be mapped from the architecture models. However, they remained in the metamodel so that they may be later added by a modeler, as discussed on section 5. This way we may have richer architectures while still being able to perform the metrics calculation.

3.2 Step 2 – Domain Metric Analysis

The metrics of [27] evaluate the overall adaptability of a system, or of one of its architecture models, based on the adaptability of each of its elements. Each element has an adaptability index (EAI) which may be either 0 - for non-adaptable elements - or 1 - for adaptable elements. An element is indistinctively a component or a connector.

The EAI value is manually assigned. The metrics themselves are defined as follows.

Architecture Adaptability Index (AAI) = EAI for all elements of the architecture model / Total number of elements of the architecture model.

Software Adaptability Index (SAI) = AAI for all architecture models of the software / Total number of architecture models for that software.

Hence, the values for these metrics vary in a range from 0 to 1, on which the higher the value the higher is the adaptability of the evaluated architecture model or of the software as a whole.

3.3 Step 3 – *i** Metrics Formulation

In this step, the metrics are translated into OCL expressions over the adapted *i** metamodel. This may be done based on the catalogue of patterns for defining *i** metrics [33]. The main patterns used in the mapping of these metrics are summarized in Table 1.

EAI is an instance of the *Property* class, and its value is assigned by an analyst. In section 5 we are going to provide some guidelines for calculating the EAI value. To calculate AAI of a software architecture, it is needed to sum (Sum pattern) the EAI value (Property-Based pattern) of each element of the architecture – i.e., each actor and each dependency (All Elements of a Kind pattern). This value will be divided by the total of elements in that architecture (Count and Normalization patterns). As a result, we have that:

context Model
$Value_{AAI}$ ::= self.allActors().eai()->sum()+self.allDependencies().eai()->sum()
$Size_{AAI}$ ::= self.allActors()->size() + self.allDependencies()->size()

Dividing $Value_{AAI}$ by $Size_{AAI}$ (Normalization pattern) the result is:

AAI ::= $Value_{AAI}$ / $Size_{AAI}$
AAI ::= self.allActors().eai()->sum()+self.allDependencies().eai()->sum() / self.allActors()->size() + self.allDependencies()->size()

on which *self* is the model under analysis.

Now, calculating SAI is just calculating the AAI of each architectural model (Sum pattern) and dividing it by the amount of architectural models (Normalization pattern):

```
SAI ::= allModels().aai()->sum() / allModels()->size()
```

Table 1. A summary of the main *i** metrics patterns used for the definition of the AAI and SAI metrics.

Name	Description	Form
Sum	The metric applied over elements of one type (aggregated) is the sum of the metric applied over the elements of the other (aggregee) that it contains	**context** *Aggregated*::metric(): *Type* **post**: result = self.*aggregees*().metric()->sum()
All Elements of a Kind	The metric counts the number of elements of a particular type that exist in a model	**context** Model::metric(): Integer **post**: result = *elem*.allInstances()->size()
Property-Based	The value that an element has for a given property is obtained	**context** Node::propertyEval(name: String): *Type* **pre**: self.value->select(v | v.property.name = name)->size() = 1 **post**: self.value->select(v | v.property.name = name).val
Normaliza-tion	The metric have a value that depend on the number of elements of a certain type	**context** *Elem*::metric(): *Type* **post**: *Size* = 0 implies result = 1.0 **post**: *Size* > 0 implies result = *Value / Size*

3.4 Step 4 – *i*MDF Update

In the definition of these metrics 7 out of a total of 24 patterns were used (23 from the original work [33] plus one added in [34]): Set of Model Elements; Measuring Instrument; Sum; Property-Based; All Elements of a Kind; Count; and Normalization. This gives a usage of 29,17%. All the steps of the metrics formalization in OCL were covered by the patterns. Therefore, no suggestion for new patterns emerged. This indicates the utility of the catalog in guiding the metrics definition.

4 Example of Application

In this section we present an application of the metrics defined in Section 3, comparing the adaptability of the three architectures presented in Fig. 4. All of these candidate architectures are for a system which is supposed to gather clinical info from a patient through a blood sensor and, if the data implies that the patient is in a critical condition, alarm someone responsible for that patient. The differences between A1

and A2, as well as those between A1 and A3, are highlighted in Fig. 4 and will be further explained.

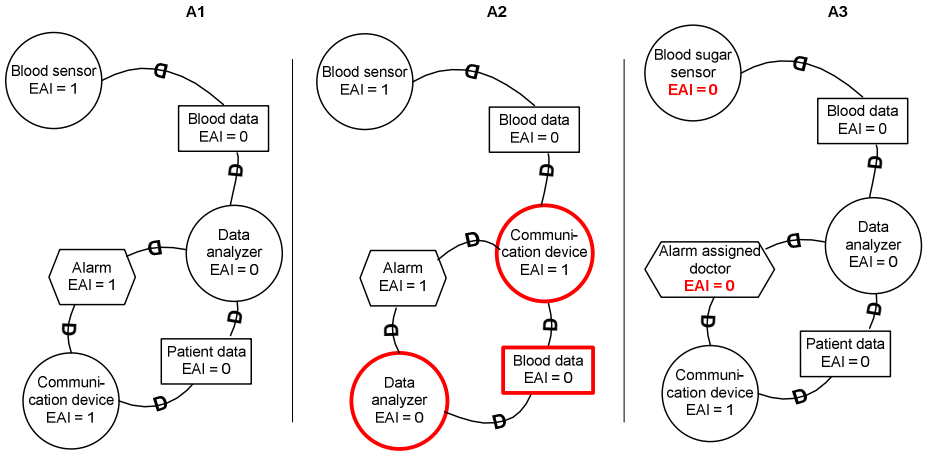

Fig. 4. An example of different candidate architectures for a health monitoring system expressed in *i** models. A1 and A2 have different structures, whilst A1 and A3 share the same structure but their elements have different adaptability index.

The A1 architecture has three adaptable elements: the Blood sensor (with respect to the blood data to be gathered), the Alarm (with respect to whom may receive the alarm) and the Communication device (with respect to the communication protocol to be used). Having three adaptable elements, out of a total of six elements, the AAI for A1 is 0.5.

A1: EAI sum (actors) = 2
A1: EAI sum (dependencies) = 1
A1: Total number of elements = 3 actors + 3 dependencies = 6
A1: AAI = 3 / 6 = 0.5

A2 differs from A1 in the sense that the Data analyzer does not get the blood data directly from the Blood sensor anymore. Instead, this data is passed through the Communication device. I.e., now the Data analyzer is a remote component. This alternative architecture may have different performance, reliability, costs, and so on, but it has the same adaptability of A1 (AAI = 0.5), as follows.

A2: EAI sum (actors) = 2
A2: EAI sum (dependencies) = 1
A2: Total number of elements = 3 actors + 3 dependencies = 6
A2: AAI = 3 / 6 = 0.5

The architecture A3 is also very similar to A1, but it differs by being a more specific one. Its elements Blood sugar sensor and Alarm assigned doctor are not adaptable (EAI = 0). Hence, the overall adaptability of A3 will be smaller than the adaptability of A1 and A2, as follows.

A3: EAI sum (actors) = 1
A3: EAI sum (dependencies) = 0
A3: Total number of elements = 3 actors + 3 dependencies = 6
A3: AAI = 1 / 6 = 0.16

5 Guidelines for defining the EAI

The EAI is a measure of an architectural element's adaptability, on which an element may be either a component or a connector. If the element is adaptable, its EAI is 1. If the element is not adaptable, its EAI is 0. The EAI value is an input to calculate the metrics presented in Section 3. In this section we are going to describe some indications of adaptability in *i** models, helping the analyst to assess the EAI for an element. This would be the case when an analyst detailed the components using *i** models after the mapping, or when the architecture was generated from *i** models [25].

In the subsection 5.1 we are going to consider only a usual version of the *i** modeling notation, based on intentional variability and on softgoals. The EAI analysis on *i** extensions for better representing information related to adaptability on goal models will be described in the subsection 5.2.

5.1 Calculating EAI in *i** models

There are three indications of component adaptability on *i** models: alternative Means-End links; Decomposition onto an Adaptability softgoal; and Contribution links to an Adaptability softgoal. The first indication of a component's adaptability is that related to intentional variability. Intentional variability is "the variability in stakeholder goals and their refinements" [1]. At the requirements level (problem space) a means-end link with more than one task (means) express that each one of these tasks is a possible way of achieving that end. However, it is yet to be defined whether only one, a subset of or all of the possible alternative means will be selected to be developed. Usually, this selection is based on the contribution of these alternatives onto softgoals.

At the architectural level (solution space) these multiple means to an end express that in the final system all of these means will be available (Fig. 5 Component A). This is an indication of adaptability, since there will be more than a single way of achieving a goal of that component. A further refinement of the alternative tasks may provide more information about its adaptability.

Another way of expressing the adaptability of a component is through the use of softgoals. The decomposition of a task into a softgoal expresses a quality constraint attached to that task. Thus, if a task is decomposed into a softgoal related to adaptability (Fig. 5 Component B), it means that the task must be adaptable. Both the task and the softgoal may be further refined, in order to define how this adaptability will be achieved. Some considerations on refining the softgoal adaptability are presented in [22].

Besides being a quality constraint for a task, adaptability may also be the softgoal of a component itself (Fig. 5 Component C). In this case, the softgoal may be further refined through contribution links, defining how it will be operationalized.

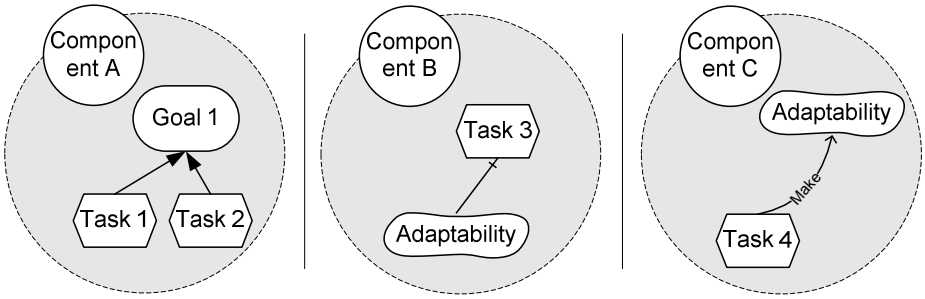

Fig. 5. Excerpts of component models presenting cases of adaptability. Component A shows its adaptability through alternative means to achieve a goal. In the Component B, the decomposition of a task into an adaptability softgoal express that the execution of this task should be adaptable. In the Component C adaptability is a softgoal that should be satisfied through contribution links.

The adaptability of a connector (dependency) is more difficult to assess since, unlike the component, a connector is not refined. However, there are two cases that are indicators of a connector's adaptability.

When the dependum of a dependency is a goal or a softgoal (Fig. 6 a) there is a margin for achieving the goal in different ways. It is up to the dependee to define how it will provide the achievement of that goal. In this sense, there may be adaptability on how the goal is achieved.

A similar case is that on which a dependency is connected to a goal or a softgoal (Fig. 6 b). Since these are abstract elements (in contrast to tasks and resources), there may be a flexibility on how to provide the dependum they require. This flexibility may be a source of adaptability.

However, in these two indicators of connector's adaptability it should be analyzed if the adaptability is of the connector itself or only of the linked components.

The cases proposed on this section may be incorporated in modeling tools to assist the analyst in assessing the element's adaptability index, providing suggested values. Specifically for the second and third cases for component's adaptability, it will be required a mechanism for identifying concepts related to adaptability, such as autonomicity and automatic changes. This identification could be performed as an ontology-based reasoning, for instance.

Since modeling is essentially a subjective activity, with the resulting model varying with the background, experience and even preferences of the modeler, the definition of the EAI is also a subjective analysis. Therefore, the cases presented above are only hints of adaptability, not aiming to be definitive rules in deciding whether an element is adaptable or not. However, there is a variety of extensions to the *i** framework and to other goal modeling notations that provide richer and more specific ways of expressing adaptability in goal models. Some of these extensions are discussed in the following subsection.

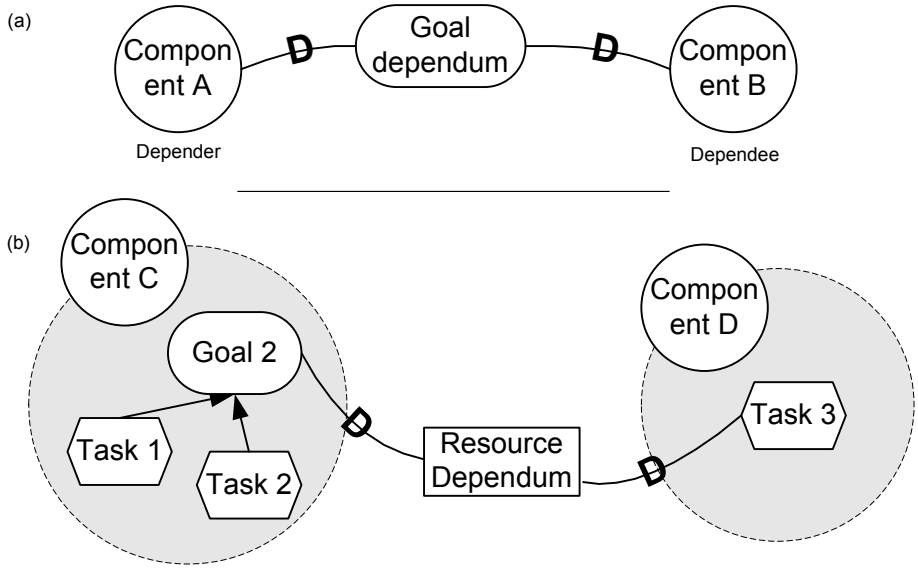

Fig. 6. Excerpts of *i** models presenting dependency links that may be adaptable. In (a) the adaptability is related to the dependum. In (b) the adaptability is related to the internal element to which the dependency is connected.

5.2 Calculating EAI in extended *i** models

Some approaches for adaptability involve the extension of goal modeling languages in order to provide more data related to monitoring and to the adaptation itself. The extra information on such models may be mapped to architectural diagrams as well, enabling the EAI assessment with more precision. In this subsection we are going to provide an overview of these approaches and then discuss how they may impact the EAI calculation through an example.

Lapouchnian and Mylopoulos [1] and Ali et al [32] use the notion of context to express domain variability. The goal model is annotated with context expressions that define conditions on the model elements. Both approaches are concerned with reasoning at requirements level, without prescribing any specific architecture.

Dalpiaz et al [14] also uses context-enriched goal models. Besides constraining the selection of alternatives, the context is used to define activation events and commitment conditions for goals and preconditions to tasks. Compensations are also defined to mitigate the occurrence of failures.

Morandini, Penserini and Perini [26] propose the development of self-adaptive systems as multi-agents systems, using goal models enriched with environmental and fault modeling. This fault modeling is similar to the concept of obstacles that is part of KAOS [6].

Some approaches are based on the notion that requirements might change at runtime and that the system should be able to respond to these changes with minimal

human intervention. Jian et al [35], Qureshi et al [28] and Bencomo et al [29] allow the insertion of goals at runtime. Each proposes different mechanisms to define how the system will satisfy those new requirements.

Baresi and Pasquale [24] propose the use of adaptive goals, in contrast to conventional goals. The adaptive goals specify countermeasures to be performed when a conventional goal is violated.

Table 2 summarizes the described approaches, allowing a comparison in respect to the extension they propose and to the architecture they are tied to, if any. In these approaches the notions of context and of environment share essentially the same meaning, as well as the notions of compensation, recovery activities and countermeasures. For the sake of uniformity, we used only the terms context and recovery activities, respectively.

Table 2. Summary of approaches for adaptability based on goal models, for quick comparison.

Approach	Base notation	Extension towards adaptability	Architecture
[1]	Tropos	Context annotations	Not defined
[32]	Tropos	Context annotations	Not defined
[14]	Tropos	Context annotations; Recovery activities	Self-reconfiguring component
[26]	Tropos	Context annotations; Fault modeling; Recovery activities	Multi-agent
[35]	*i**	Changing the model at runtime; Context annotations	Not defined
[28]	Techne	Changing the model at runtime;	Service-based
[29]	KAOS	Changing the model at runtime; Flexibility language	Not defined
[24]	KAOS	Recovery activitites	Service-based

The extra information of these enriched models, if also present on the architecture model, e.g. considering a component with context annotations (Fig. 7), can help with the assessment of EAI. A simple task decomposition would not be enough to state that this component is adaptable. However, the context annotation means that: if the context is true, Task 1 is decomposed on Task 2 and Task 3; if the context is false, Task 1 is decomposed only on Task 3. This is a strong indication of adaptability.

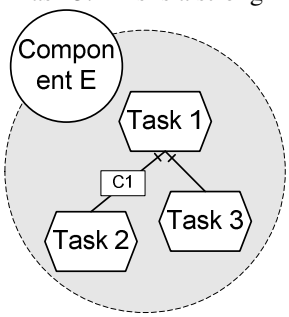

Fig. 7. Excerpt of an *i** model presenting a task-decomposition with context annotation. Task 2 is a decomposition of Task 1 if and only if the context C1 is true.

A similar analysis can be performed with other extensions: if the sub-model that represents an element may be changed at runtime, than this element can be considered adaptable; if there are recovery activities associated with an element, than this element may also be considered adaptable. Nonetheless, further work would be required to define more precise indicators for each extension.

6 Conclusion and Future Works

In this paper we presented the derivation of two architectural adaptability metrics into *i** adaptability metrics, using the *i*MDF$_M$ method. With the *i** expressiveness it is possible to identify the adaptability of individual elements, which is an input for the metrics. These metrics allow the comparison of different *i** models in order to decide which model is more adaptable. The work aims to stress the fundamental role that metrics may play in the context of adaptive systems, for deciding when, and towards where, adapt. We align with the opinion that in order to perform adaptation as automatic as possible, it is necessary to have available a range of metrics that support making informed decisions.

As a side contribution, this work can be considered a new exemplar in the consolidation of the *i*MDF framework. We have checked the high-degree of completeness of the current metric patterns catalogue, and the adequacy of the *i*MDF$_M$ method for facilitating the mapping of a metric defined over a particular domain (here, software architectures) onto *i**.

Future work includes defining mechanisms for automatically defining the Element Adaptability Index (EAI), making use of the expressiveness of *i** models. Then, we may evolve the metrics to consider different degrees of adaptability on individual elements, enabling a more detailed comparison. More evidence would be required to assess these enhanced metrics. Further, we intend to develop heuristics for changing architectures represented in *i** models towards more adaptable systems. The resulting architectures would be compared using the metrics here defined.

Acknowledgements

This work has been partially supported by the Spanish research project TIN2007-64753, UPV PAID-02-10, Erasmus Mundus External Cooperation Window - Lot 15 Brasil and the Brazilian institutions CAPES and CNPq.

References

1. Alexei Lapouchnian, John Mylopoulos: Modeling Domain Variability in Requirements Engineering with Contexts. ER 2009: 115-130.
2. Alexei Lapouchnian, Yves Lespérance: From Adaptive Systems Design to Autonomous Agent Design. *i** Workshop, 2010: 108-112.

3. Alexei Lapouchnian. Goal-oriented requirements engineering: An overview of the current research. Technical Report http://www.cs.toronto.edu/~alexei/pub/Lapouchnian-Depth.pdf, University of Toronto, 2005.
4. Axel van Lamsweerde: From System Goals to Software Architecture. SFM 2003: 25-43.
5. Axel van Lamsweerde: Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice. RE 2004: 4-7.
6. Axel van Lamsweerde: Requirements engineering: from system goals to UML models to software specifications. Wiley, 2009.
7. Betty Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, others: Software engineering for self-adaptive systems: A research roadmap. Software Engineering for Self-Adaptive Systems - LNCS 5525/2009:1-26.
8. Carla T. L. L. Silva, Jaelson Castro, Patricia Azevedo Tedesco, João Araújo, Ana Moreira, John Mylopoulos: Improving Multi-Agent Architectural Design. SELMAS 2006: 165-184.
9. Dilip Soni, Robert L. Nord, Christine Hofmeister: Software Architecture in Industrial Applications. ICSE 1995: 196-207.
10. Emanuel Santos: A Proposal of Metrics for Evaluating *i** Models (in Portuguese: Uma Proposta de Métricas para Avaliar Modelos *i**). MSc Dissertation, Univ. Federal de Pernambuco (2008).
11. Eric Yu, Jaelson Castro, Anna Perini: Strategic Actors Modeling with *i**. Tutorial at RE 2008.
12. Eric Yu, John Mylopoulos: Why Goal-Oriented Requirements Engineering. REFSQ 1998: 15-22.
13. Eric Yu: Modelling Strategic Relationships for Process Reengineering. PhD Dissertation, Univ. of Toronto (1995).
14. Fabiano Dalpiaz, Paolo Giorgini, John Mylopoulos: An Architecture for Requirements-Driven Self-reconfiguration. CAiSE 2009: 246-260.
15. Fernanda M. R. Alencar, Beatriz Marín, Giovanni Giachetti, Oscar Pastor, Jaelson Castro, João Henrique Pimentel: From *i** Requirements Models to Conceptual Models of a Model Driven Development Process. PoEM 2009: 99-114.
16. Fernanda M. R. Alencar, Jaelson Castro, Márcia Lucena, Emanuel Santos, Carla T. L. L. Silva, João Araújo, Ana Moreira: Towards modular *i** models. SAC 2010: 292-297.
17. Gemma Grau, Xavier Franch: On the Adequacy of *i** Models for Representing and Analyzing Software Architectures. ER Workshops 2007: 296-305.
18. Ivan Jureta, John Mylopoulos, Stéphane Faulkner: A core ontology for requirements. Applied Ontology 4(3-4): 169-244 (2009).
19. Jaelson Castro, Xavier Franch, John Mylopoulos, E. Yu (eds.). Fourth International i* Workshop (*i**'10). CEUR Workshop Proceedings 586, June 2010.
20. John Mylopoulos, Jaelson Castro, Manuel Kolp: Tropos: Toward agent-oriented information systems engineering. AOIS2000, June 2000.
21. Kristopher Welsh, Pete Sawyer: Deriving Adaptive Behaviour from *i** Models. *i** Workshop 2010: 98-102.
22. Lawrence Chung, Kendra Cooper, Anna Yi: Developing Adaptable Software Architectures Using Design Patterns: an NFR Approach. Journal of Computer Standards & Interfaces 25(3): 253-260 (2003).
23. Lúcia R. D. Bastos, Jaelson Castro, John Mylopoulos: Deriving Architectures from Requirements. RE 2006: 332-333.
24. Luciano Baresi, Liliana Pasquale: Live Goals for Adaptive Service Compositions. SEAMS 2010:114-123.
25. Márcia Lucena, Jaelson Castro, Carla T. L. L. Silva, Fernanda M. R. Alencar, Emanuel Santos, João Pimentel: A Model Transformation Approach to Derive Architectural Models from Goal-Oriented Requirements Models. OTM Workshops 2009: 370-380.

26. Mirko Morandini, Loris Penserini, Anna Perini: Automated Mapping from Goal Models to Self-Adaptive Systems. ASE 2008: 485-486.
27. Nary Submaranian, Lawrence Chung: Metrics for Software Adaptability. Software Quality Management 2001.
28. Nauman Qureshi, Anna Perini, Neil Ernst, John Mylopoulos: Towards a Continuous Requirements Engineering Framework for Self-Adaptive Systems. 1st International Workshop on requirements at run-time 2010.
29. Nelly Bencomo, Jon Whittle, Peter Sawyer, Anthony Finkelstein, Emmanuel Letier: Requirements reflection: requirements as runtime entities. ICSE 2010: 199-202.
30. Philippe Kruchten. The 4+1 View Model of Architecture. IEEE Software 12(6): 42-50 (1995).
31. Raian Ali, Amit K. Chopra, Fabiano Dalpiaz, Paolo Giorgini, John Mylopoulos, Vitor Souza: The Evolution of Tropos: Contexts, Commitments and Adaptivity. *i** Workshop 2010:15-19.
32. Raian Ali, Fabiano Dalpiaz, Paolo Giorgini: A Goal Modeling Framework for Self-contextualizable Software. BMMDS/EMMSAD 2009: 326-338.
33. Xavier Franch, Gemma Grau: Towards a Catalogue of Patterns for Defining Metrics over *i** Models. CAiSE 2008: 197-212.
34. Xavier Franch: A Method for the Definition of Metrics over *i** Models. CAiSE 2009: 201-215.
35. YunSong Jian, Tong Li, Lin Liu, Eric Yu: Goal-Oriented Requirements Modelling for Running Systems. 1st International Workshop on requirements at run-time 2010.