iStarML.
The * Mark-up Language

REFERENCE’S GUIDE

Carlos Cares$^{1,2}$ ccares@lsi.upc.edu
Xavier Franch$^1$ franch@lsi.upc.edu
Anna Perini$^3$ perini@itc.it
Angelo Susi$^3$ susi@itc.it

1 Technical University of Catalonia, C/Jordi Girona, 1-3, 08034
Barcelona, Spain
2 University of La Frontera, Avenida Francisco Salazar 01145,
Temuco, Chile
3 ITC-irst, Trentine Culture Institute, Scientific and Technological
Research Centre, 38050 Povo, Trento, Italy

September 2007
Contents

1 Introduction ...................................................................................................................... 3
2 Syntax Expression ......................................................................................................... 5
4 Representing Actors .................................................................................................... 7
5 Representing Intentional Elements ............................................................................. 8
6 Representing Actor’s boundaries ................................................................................ 9
7 Representing Actor’s Rationale .................................................................................. 10
8 Representing Dependencies ....................................................................................... 14
9 Representing actor’s relationships ............................................................................. 19
10 iStarML’s Graphic specification .............................................................................. 22
Conclusions ..................................................................................................................... 28
Appendix A. Complete code of example 10.6 ............................................................... 29
References ....................................................................................................................... 36
1 Introduction

iStarML is an XML compliant format [1] to represent $i^*$ diagrams [2]. Therefore it is a textual specification. It is not the aim of this document neither to standardize the semantic of $i^*$ nor its graphic expression. Besides, the syntax specification could generate structures which do not have any particular semantic interpretation.

Different methodologies have been created based on $i^*$ concepts and modelling techniques. In particular the $i^*$ framework has been exploited in different areas such as organizational modelling, business process reengineering and requirements engineering. Moreover, some proposals have been made to incorporate $i^*$ modelling concepts to deal with software systems requirements representation and design. An example of these proposals is Tropos [3, 4], an agent-oriented software development methodology. The contribution of Tropos at the requirements stage and in agent-oriented design has been acknowledged by different comparative studies [5-7]. Also relevant is GRL [8], an $i^*$ variation which has been added as part of the industrial Telecommunications Standard Z150 [9] for systems specification. Besides these three proposals: $i^*$, Tropos and GRL, there are also other ones that have introduced several constructs in the language with different research aims, such as security and trust concerns [10-12], temporal operators [13], and traceability constructs [14], among others.

Therefore, the goal of iStarML is to have a common format where the common conceptual framework of the main $i^*$ language variations is made explicit and, in addition, the differences could be expressed using open options using the same specification.

Consequently a common representation of $i^*$ diagrams allow:

1. To have a file format for diagrams interchanging among different type of specific $i^*$ software tools such as goal-analysis, designing, editors, metric calculation, etc.
2. To have a common way of representing the differences and similarities among the existing $i^*$ variations.
3. To have a common representation for repository of $i^*$ patterns
4. To take advantages of the XML format for Internet communication and also the use of general XML tools.

The main iStarML set of tags corresponds to the abstract set of core concepts which are part of the seminal proposal [2, 15] and also they are present on a broad set of related proposals [4, 8, 10-13, 16-18]. The defined core concepts and its tags are showed on table 1.1. In order to provide additional features there are especial tags which are not part of any related proposal of $i^*$. It has been included with topics related the use of XML in a context of storing and recovering $i^*$ diagrams. These are presented on table 1.2.
Table 1.1 Core concepts of i*-based modelling languages and the corresponding iStarML tags

<table>
<thead>
<tr>
<th>Abstract core concept</th>
<th>Meanings and examples of core specializations</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>An actor represents an entity which may be an organization, a unit of an organization, a single human or an autonomous piece of software. Also it can represent abstractions over actors such as roles and positions.</td>
<td>&lt;actor&gt;</td>
</tr>
<tr>
<td>Intentional element</td>
<td>An intentional element is an entity which allows to relate different actors conforming a social network or, also, to express the internal rationality of an actor. Broadly used types of intentional elements are: goal, softgoal, resource, and task.</td>
<td>&lt;ielement&gt;</td>
</tr>
<tr>
<td>Dependency</td>
<td>A dependency is a relationship which represents the explicit dependency of an actor (dependee) respect to the other actor (dependee). The dependency is expressed with respect to an intentional element.</td>
<td>&lt;dependency&gt; &lt;dependee&gt;</td>
</tr>
<tr>
<td>Boundary</td>
<td>A boundary represents a group of intentional elements. The common type of boundary is the actor’s boundary which represents the vision of an omnipresent objective observer with respect to the actor’s scope. However other boundary types can also be used.</td>
<td>&lt;boundary&gt;</td>
</tr>
<tr>
<td>Intentional element link</td>
<td>An intentional element link represents an n-ary relationship among intentional elements (either in the actor’s boundary or outside). Broadly used types of intentional element link are decomposition, means-end and contribution. Related concepts such as routines or capabilities can be also represented using this relationship</td>
<td>&lt;ielementLink&gt;</td>
</tr>
<tr>
<td>Actor association link</td>
<td>An actor relationship is a relationship between two actors. Broadly used types of actor relationships are is_a, is_part_of, instance_of (INS), plays, occupies and covers.</td>
<td>&lt;actorLink&gt;</td>
</tr>
</tbody>
</table>
Table 1.2 Complementary iStarML tags

<table>
<thead>
<tr>
<th>Additional Concept</th>
<th>Tag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>i* markup language file</td>
<td>&lt;istarml&gt;</td>
<td>The main tag of the iStarML</td>
</tr>
<tr>
<td>Diagram</td>
<td>&lt;diagram&gt;</td>
<td>A diagram is a particular i* diagram</td>
</tr>
<tr>
<td>Graphic expression</td>
<td>&lt;graphic&gt;</td>
<td>Represent some graphic properties of a particular diagram or diagram element.</td>
</tr>
</tbody>
</table>

The extensibility of the iSTarML proposal is provided by allowing additional XML attributes on the static set of iStarML tags. This option seems to be the best one in order to keep a closed core set of fundamental concepts, which would allow managing the attribute-based extensionality because the corresponding semantic is mainly associated to the core concept in place of their attributes.

2 Syntax Expression

In order to express the syntactical options we will use the traditional extended BNF meta language [19]. However, given the characters “<” and “>” are part of the language, it is not possible for them to be part of the meta language. We have omitted them but we have marked the defined elements using the color blue and the italic style. The meta symbols definition is showed in table 2.1

Table 2.1 Used extended BNF symbols

<table>
<thead>
<tr>
<th>Italic blue string</th>
<th>means a language concept (in place of the traditional BNF symbols “&lt;” and “&gt;”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>::=</td>
<td>means a language definition</td>
</tr>
<tr>
<td>[ ]</td>
<td>means an optional language structure, 0 or 1 time</td>
</tr>
<tr>
<td>( )</td>
<td>means that a language structure could be repeated 0 or more times</td>
</tr>
<tr>
<td>( )</td>
<td>group of language structures</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some italic blue symbols are considered terminal symbols when they are referred to traditional data types such as integer, real or string. Another non-defined data type is the `hexrgbcolor` type, which is used to represent a RGB hexadecimal colour e.g. 0000FF to represent a pure blue.
A BNF can not express some specific language features like the requirements that a reference exists in some place of the same file. In iStarML we use two attributes which require a string value which appears like the unique value assigned to the xml’s tag identifier, i.e. the id attribute. These values are iref and aref. The first one requires a string value which has been used only one time like the id attribute value of an ielement tag (defined in section 5). The second one, the aref value, requires a string value which has been used only one time like the id attribute value of an actor tag (defined in section 4). Given that these values have an especial the described especial meaning in the BNF specification it is used also the blue color, but they have the above definition. Also it is used some blue color for describing another known data types likes integer and string which have the traditional definitions.

3 Basic Structure of the iStarML format

The tag <istarml> is the main tag of iStarML. It can content only the <diagram> tag. In the table 3.1 we show the options of this tag. Under this structure it is possible to store on the same file a set of different i* diagrams.

Table 3.1 <istarml> syntax

| istarmlFile ::= | <istarml version="1.0"> diagramTag { diagramTag } |</istarml> |
| diagramTag ::= | <diagram basicAtts [author=string] [extraAtt] > [ graphic-diagram] { [actorTag] [ielementExTag] } |</diagram> |
| extraAtt ::= | attributeName = attributeValue |
| basicAtts ::= | [id="string"] name="string" [id="string" [name=" string"] |

Example 3.1 Basic structure of an iStarlML file

```xml
<istarml version="1.0">
  <diagram>
    <diagram>
    </diagram>
  </diagram>  
  <diagram>
  </diagram>
</istarml>
```
4 Representing Actors

For representing actors it has been defined the actor tag. The BNF of table 4.1 shows the syntactic alternatives of this tag. Mainly the different types of actor can be handled by using the type attribute. The example 4.1 illustrates a basic use of the tag for representing two actors. The use of additional options of the actor tag is explained in the context of the boundary tag (section 6) and the representations of intentional relationships (section 7).

Table 4.1 <actor> syntax.

| actorTag ::= | <actor basicAtts [typeAtt] [extraAtt] > |
|             | [graphic-node] [actorLinkTag] [boundaryTag] |
|             | </actor> |
|             | <actor basicAtts [typeAtt] [extraAtt] /> |
|             | <actor aref="string" /> |
|             | <actor aref="string"> [graphic-node] </actor> |
| typeAtt ::= | type="actorType" |
| actorType ::= | basicActorType \ string |
| basicActorType ::= | agent | role | position |

Example 4.1 Basic representation of two actors

```xml
<?xml version="1.0"?>
<lstaml version="1.0">
  <diagram name="Example 4.1">
    <actor name="Hospital"/>
    <actor name="Patient" type="role"/>
  </diagram>
</lstaml>
```
5 Representing Intentional Elements

An intentional element is an abstraction over a set of different i*’s constructs such as goal, softgoal, resource or task. Some i*’s variations considers additional types of intentional elements such as belief [8] or constraint [18]. The iStarML proposal considers all these kinds of intentional elements which can be represented using the \texttt{ielement} tag. The syntax is specified in Table 5.1.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Table 5.1} \texttt{ielement} syntax \tabularnewline \hline
\end{tabular}
\end{table}

Example 5.1 Basic representation of intentional elements
The use of the other options of intentional’s representation is explained in the context of the boundary tag (section 6) and intentional link representations (section 7).

6 Representing Actor’s boundaries

A boundary tag represents the internal state of an actor, thus this state is represented in a nested structure inside the scope of an actor which has been also named boundary. The defined syntax is showed in table 6.1.

Table 6.1 <boundary> syntax.

| boundaryTag ::= | <boundary [type="string"]> |
|               | [graphic-path] [ielementTag] [actorTag] |
|               | </boundary> |

Example 6.1 A basic representation of an actor’s boundary
Example 6.2 Differencing internal and external elements, example taken from [18, 20].

7 Representing Actor's Rationale

The actor's rationale is given by the multiple relationships which are established among intentional elements either belonging to its boundary or outside of it. Therefore the way of representing this "rationality" is by setting the relationships which involves the intentional elements in the scope of its boundary. The tag for stating these relationships is the ielementLink tag. Its syntax is specified in table 7.1.
Table 7.1 `<iel elementLink>` syntax

```
<iel elementLink linkAtts>
[graphic-path]  ielementTag ielementTag
</iel elementLink>
```

```
lkAtts ::= type = “decomposition” [value=(“and” | “or” )] |
type=“means-end” [value=“string”] |
type=“contribution” [value=“contribution-value”] |
type=“string” [value=“string”]
```

```
contribution-value ::= + | - | sup | sub | ++ | -- | break | hurt | some- | some+ |
unknown | equal | help | make | and | or
```

Example 7.1 Tropos’s task decomposition [21]
Example 7.2 Implementing “why” as intentional relationship

Example 7.3 Representing elements from Secure Tropos [10, 22]
Representing Dependencies

Dependencies is one of the classical i*-s constructs and its aim is representing intentional relationships between two (or occasionally more) actors. To feature this relationship a specific intentional element makes the link among the involved actors which are named dependers or dependees. It represents that some actors hazard the accomplishment of its intentions (dependers) on third actors (dependees). For representing this especial kind of relationships iStarML provides the tags dependency, depender and dependee. The specific syntax is showed in table 8.1.

This language construct is designated to consider the intentional element that gives the meaning to the dependency and thus it plays the central role in the dependency specification. Therefore the dependency is built like a nested structure from an intentional element. This situation means that actors are specified only by referencing actors, either they have been already created or will appear next on the iStarML file. All the examples of this section illustrate the case.

Table 8.1 <dependency> syntax.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dependencyTag ::= &lt;dependency&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code> dependerTag { dependerTag }</code></td>
<td></td>
</tr>
<tr>
<td><code>[ dependeeTag ]</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;/dependency&gt;</code></td>
<td></td>
</tr>
<tr>
<td>`dependerTag ::= &lt;depender [iref=&quot;string&quot;] aref=&quot;string&quot; [value=&quot;dep-type&quot;] /&gt;</td>
<td>l`</td>
</tr>
<tr>
<td><code>&lt;depender [iref=&quot;string&quot;] aref=&quot;string&quot; [value=&quot;dep-type&quot;] &gt; [graphic-path] &lt;/depender&gt;</code></td>
<td></td>
</tr>
<tr>
<td>`dependeeTag ::= &lt;dependee [iref=&quot;string&quot;] aref=&quot;string&quot; [value=&quot;dep-type&quot;] /&gt;</td>
<td>l`</td>
</tr>
<tr>
<td><code>&lt;dependee [iref=&quot;string&quot;] aref=&quot;string&quot; [value=&quot;dep-type&quot;] &gt; [graphic-path] &lt;/dependee&gt;</code></td>
<td></td>
</tr>
<tr>
<td>`Dep-type ::= open</td>
<td>committed</td>
</tr>
</tbody>
</table>
Example 8.1 Basic representation of dependency

```
<?xml version="1.0"?>
<starml version="1.0">
  <diagram name="Example 0.1">
    <actor id="21" name="A"/>
    <actor id="10" name="B"/>
    <element type="goal" name="G">
      <dependency>
        <dependee ref="21"/>
        <dependee ref="10"/>
      </dependency>
    </element>
  </diagram>
</starml>
```

Example 8.2 Dependency from an internal intentional element

```
<?xml version="1.0"?>
<starml version="1.0">
  <diagram name="Example 0.2">
    <actor id="201" name="A">
      <boundary>
        <element id="101" type="task" name="T1"/>
      </boundary>
    </actor>
    <actor id="230" name="B"/>
    <element name="G" type="goal">
      <dependency>
        <dependee ref="101" ref="201"/>
        <dependee ref="230"/>
      </dependency>
    </element>
  </diagram>
</starml>
```
Example 8.3 Dependency from a nested actor to multiple dependees

```xml
<?xml version="1.0"?>
<istxml version="1.0">
<diagram name="Example 8.3">
  <actor id="201" name="A">
    <boundary>
      <actor id="20" name="W">
        <boundary>
          <element id="101" type="task" name="T1"/>
        </boundary>
      </actor>
    </boundary>
  </actor>
</diagram>
</istxml>
```

Example 8.4 Extended dependencies from Secure Tropos [10, 22]
Example 8.5 Abstract self dependency taken from Tropos-PL [23]
Example 8.6 Representing the owner relationship from Secure Tropos [22]
9 Representing actor’s relationships

Actors’ relationships are present in most of the i* variations and, in all cases, they are asymmetric relationships, i.e., if A and B are related actors under the relationship R, then generally, B is not related with A under R. Traditional actors’ relationships are: is_part_of, is_a, plays, occupies and covers. However these do not constitute a complete set. In order to get an abstraction of all these relationships the tag actorLink, is the construct designed for specifying these actors’ relationships, the attribute type can be used to specify the relationship. The syntax is specified in table 9.1.

Table 9.1 <actorLink> syntax

| actorLinkTag ::= | <actorLink type="actorLink-type" aref="string"> |
| graphic-path ] </actorLink> | |
| <actorLink type="actorLink-type" aref="string"/> |
| actorLink-type ::= | is_part_of | is_a | instance_of | plays | covers | occupies |
| string |

Example 9.1 Representing instance_of (INS) and is_a relationships

![Diagram showing relationships between actor entities Ag1, Ag2, and W. Ag2 is_a Ag1 and Ag1 has an instance_of relationship with W.](image-url)
Example 9.2  The two representations for is_part_of relationship

a) Using <actorLink>

```xml
<?xml version="1.0"?>
<lxmlml version="1.0">
  <diagram name="Example 9.2a">
    <actor id="201" name="University"/>
    <actor id="202" name="Engineering Faculty"/>
    <actor id="203" name="Dept of Computer Science"/>
    <actorLink type="is_part_of" ref="201"/>
    <actorLink type="is_part_of" ref="202"/>
  </diagram>
</lxmlml>
```
b) Using nested structures

```xml
<?xml version="1.0"?>
<starml version="1.0">
  <diagram name="Example 9.2b">
    <actor name="University">
      <boundary>
        <actor name="Engineering Faculty">
          <boundary>
            <actor name="Dept of Computer Science">
              <boundary>
                <factor>
                  <boundary>
                    </actor>
                  </diagram>
                </actor>
              </boundary>
            </actor>
          </boundary>
        </actor>
      </boundary>
    </actor>
  </diagram>
</starml>
```
The possibility of a graphic specification of i* elements is provided. The aim is to offer the graphic information which allows having a general map of the distribution of the graphic elements on the plane. Therefore we have defined a basic syntax for a graphic specification where, the specific shapes of the intentional elements and actors are not specified. However the shape of the actors’ boundary and the path of the link connections could be declared using a set of graphic options.

Additionally, we are also consider the XML-based graphic proposal namely Scalar Vector Graphic (SVG) [24]. This proposal constitutes a contemporary way of representing graphic information and, moreover, there are several initiatives which provides of end-user applications and software development tools, such as editors, parsers and browsers among others [25].

Therefore, we account with two alternative ways of specifying graphic expressions. Both are present in our EBNF specification showed at table 10.1.

Table 10.1 <graphic> syntax

| graphic-diagram ::= | <graphic content="SVG"> svg-content </graphic> |<graphic content="basic" g-options-diagram /> | |
| g-options-diagram ::= | xpos="number" ypos="number" width="number" height="number" [unit="unit"] [bgcolor="hexrgbcolor"] |
| graphic-node ::= | <graphic content="SVG"> svg-content </graphic> |<graphic content="basic" g-options-node /> | |
| g-options-node ::= | xpos="number" ypos="number" width="number" height="number" [unit="unit"] [bgcolor="hexrgbcolor"] [fontcolor="hexrgbcolor"] [fontfamily="string"] [fontsize="number"] |
| unit ::= | cm | in | pt |
Example 10.1 Basic coordinates in graphic representations

```xml
<diagram name="Example 10.1">
  <sector name="Student">
    <graphic content="basic" xpos="2.1" ypos="1.53" unit="cm">
      <point xpos="number" ypos="number" />
      <point xpos="number" ypos="number" />
      [ <point xpos="number" ypos="number" /> ]
    </graphic>
  </sector>
</diagram>
```
Example 10.2 Combining graphic tags to represent a complete diagram

```xml
<example name="Example 10.2">
  <figure>
    <div>
      <xml>
        <bml version="1.0">
          <diagram name="Example 10.2">
            <actor id="A11" name="A"/>
            <actor id="A12" name="B"/>
            <boundary>
              <element id="AI" type="resource" name="R">
                <graphic content="basic" xpos="0.7" ypos="4.43" unit="cm"/>
                <point ypos="0.78"/>
              </element>
            </boundary>
            <actor>
              <element name="G" type="goal">
                <graphic content="basic" xpos="2.7" ypos="0.52" unit="cm"/>
                <dependency>
                  <dependency id="A11"/>
                  <graphic content="basic" shape="spline" unit="cm">
                    <point x="1.53" y="2.4"/>
                    <point x="2.37" y="2.65"/>
                  </graphic>
                </dependency>
              </element>
            </actor>
          </diagram>
        </bml>
      </xml>
    </div>
  </figure>
</example>
```
The way of using SVG in an istarml file is by embedding the istarml’s graphic tag `<graphic>` and, inside it, using proper SVG tags. Thus it is possible to keep the i* semantic information just omitting all the graphic tags and their content. On the other hand, it is possible to have a graphic representation putting together the different graphic contents of the istarml file.

To keep this specification as simple as possible, we do not go deep in to the SVG specification; however we illustrate its use by showing some basic examples.

**Example 10.3 Basic graphic properties of an i* diagram**

```
<diagram name="My i* diagram">
  <graphic content="SVG">
    <svg width="14cm" height="4cm" viewBox="0 0 1200 500">
      <text x="20" y="30" font-family="Verdana" font-size="22" fill="blue">
        My i* diagram
      </text>
    </svg>
  </graphic>
</diagram>
```

**Example 10.4 Graphic display of the title of an i* diagram using SVG**

```
<diagram name="My i* diagram">
  <graphic content="SVG">
    <svg width="14cm" height="4cm" viewBox="0 0 1200 500">
      <text x="20" y="30" font-family="Verdana" font-size="22" fill="blue">
        My i* diagram
      </text>
    </svg>
  </graphic>
</diagram>
```

**Example 10.5 Intentional element with an SVG graphic representation**

```
<element name="Protect my privacy" type="softgoal">
  <graphic content="SVG">
    <g>
      <text x="100" y="210" font-family="Verdana" font-size="30" fill="blue">
        Protect my privacy
      </text>
    </g>
  </graphic>
</element>
```
Example 10.6 A portion of the diagram extracted from [26] and its iStarML code
Conclusions

iStarML is a XML-based specification which has been presented using the traditional meta-language in Computer Science named EBNF. This specification has been built taking in consideration different meta models of the i* constructs. The derivation of the iStarML tags from the i* core concepts has allowed keeping the language simple and, at the same time, to consider different language variations using the same language constructs. For this reason we often open the original set of i* options adding any string value such a possible well formed value. However, this choice also allows making strict derivations of iStarML in order to accept only specific variation of i*.

To implement some parsing services it is possible to use different technologies such XSD, DTD or even XMI. However, the idea of implementing a non-heavy and fast specific parser also can be considered.

Moreover, there are some specific situations on the language which are new or implicit in the context of the defined i* constructs. iStarML adds and implements the concept of diagram and also it deals with the graphic distribution of the elements in a diagram. Moreover it is possible to have common elements among different diagrams, although these common elements, in this version, are restricted to the actor and ielement tags.

We really hope that this work will be a contribution to the interoperability of the i* scientific and industrial community. Therefore we are very open to push new initiatives to walk for the way of improving this approach or developing some iStarML supporting tool. Any comment, ask for or suggestion will be very welcome.
Appendix A. Complete code of example 10.6
<element type="softpar" name="Proceed Efficiently"/>
<graphic content="SVG"/>
<element type="softpar" name="Deliver gold nuggets into the depot"/>
<graphic content="SVG"/>
<element>
  <boundary>
    <actor>
      <diagram>
        <AStarml>
  </element>
References


