iStarML
The *i** 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

August 2007

Contents

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

Abstract core concept	Meanings and examples of core specializations	Tag
Actor	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.	<actor>
Intentional element	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.	<ielement>
Dependency	A dependency is a relationship which represents the explicit dependency of an actor (depender) respect to the other actor (dependee). The dependency is expressed with respect to an intentional element.	<dependency> <dependee> <depender>
Boundary	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.	<boundary>
Intentional element link	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	<ielementLink>
Actor association link	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.	<actorLink>

Table 1.2 Complementary iStarML tags

Additional Concept	Tag	Meaning
*i** markup language file	<istarml>	The main tag of the iStarML
Diagram	<diagram>	A diagram is a particular i* diagram
Graphic expression	<graphic>	Represent some graphic properties of a particular diagram or diagram element.

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

Italic blue string	means a language concept (in place of the traditional BNF symbols "<" and ">")
::=	means a language definition
[ ]	means an optional language structure, 0 or 1 time
{ }	means that a language structure could be repeated 0 or more times
( )	group of language structures
|	means options' separation

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 constructs which require that the string value appear like the value assigned to the xml's identifier tag, i.e. the attribute id. These values are *iref* and *aref*. The first one requires a string value which has been used only one time like the id value of an *ielement* tag (section 5). The *aref* value requires a string value which has been used only one time like the id value of an *actor* tag (section 4).

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

```
<istarml version="1.0">
        <diagram>
        </diagram>
        <diagram>
        </diagram>
             .
             .
             .
</istarml>
```

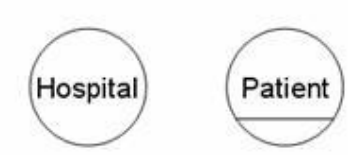
4 Representing Actors

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




```
1  <?xml version="1.0"?>
2  <istarml version="1.0">
3    <diagram name="Example 4.1">
4      <actor name="Hospital"/>
5      <actor name="Patient" type="role"/>
6    </diagram>
7  </istarml>
```

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 kind of intentional elements which can be represented using the ielement tag. The syntax is specified in table 5.1.

Table 5.1 <ielement> syntax

ielementTag ::=	<ielement *ieAtts*> [*graphic-node*]
	{*ielementLinkTag*} </ielement> |
	<ielement *ieAtts*/> |
	<ielement iref="*string*"/> |
	<ielement iref="*string*"> [*graphic-node*] </ielement>
ielementExTag ::=	<ielement *ieAtts*>
	[*graphic-node*] [*dependencyTag*]
	{*ielementLinkTag*} </ielement> |
	ielementTag
ieAtts ::=	*basicAtts* type="*itype*" [state="*istate*"] {*extraAtt*}
itype ::=	*basic-itype* | *string*
basic-itype ::=	goal | softgoal | task | resource
istate ::=	undecided | satisfied | weakly satisfied | denied |
	weakly denied | *string*

Example 5.1 Basic representation of intentional elements

```
1   <?xml version="1.0"?>
2   <istarml version="1.0">
3     <diagram name="Example 5.1">
4       <ielement type="goal" name="Supervise students' career"/>
5       <ielement type="task" name="Send personal email"/>
6     </diagram>
7   </istarml>
```

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).

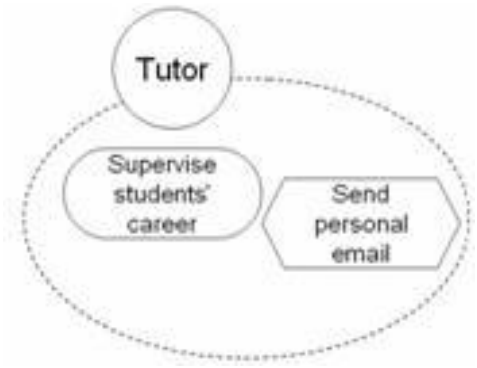
6 Representing Actor's boundaries

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



```
1   <?xml version="1.0"?>
2   <istarml version="1.0">
3     <diagram name="Example 6.1">
4       <actor name="Tutor">
5         <boundary>
6           <ielement type="goal" name="Supervise students' career"/>
7           <ielement type="task" name="Send personal email"/>
8         </boundary>
9       </actor>
10    </diagram>
11  </istarml>
```

Example 6.2 Differencing internal and external ielements, example taken from [18, 20].


```
1   <?xml version="1.0"?>
2   <istarml version="1.0">
3     <diagram name="Example 11">
4     <ielement type="softgoal" name="Protect my privacy"/>
5       <actor name="Electronic Record Mng System">
6         <boundary>
7           <ielement type="softgoal" name="Provide process performance"/>
8           <ielement type="constraint" name="Daily updated"/>
9         </boundary>
10      </actor>
11      </diagram>
12  </istarml>
```

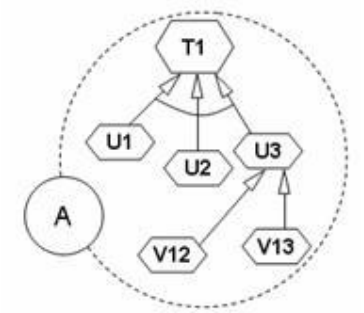
7 Representing Actor's Rationale

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 <ielementLink> syntax

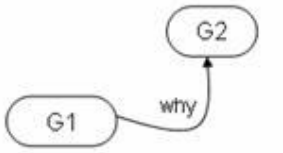
ielementLinkTag ::=	<ielementLink *linkAtts*> [*graphic-path*] *ielementTag* {*ielementTag*} </ielementLink>
linkAtts ::=	type = “decomposition” [value=(“and” | “or”)] | type=“means-end” [value=“*string*”] | type=“contribution” [value=“*contribution-value*“] | type=“*string*” [value=“*string*”]
contribution-value ::=	+ | - | sup | sub | ++ | -- | break | hurt | some- | some+ | unknown | equal | help | make | and | or

Example 7.1 Tropos’s task decomposition [21]



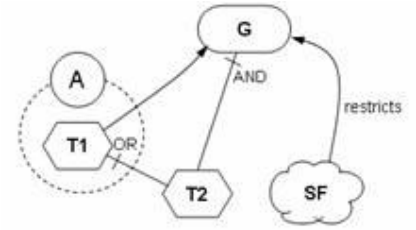
```
<?xml version="1.0"?>
<istarml version="1.0">
  <diagram name='Example 7.1'>
    <actor name="A">
      <boundary>
        <ielement name="T1" type="task">
          <ielementLink type="decomposition" value="and">
            <ielement name="U1" type="task"/>
            <ielement name="U2" type="task"/>
            <ielement name="U3" type="task">
              <ielementLink type="decomposition" value="or">
                <ielement name="V12" type="task"/>
                <ielement name="V13" type="task"/>
              </ielementLink>
            </ielement>
          </ielementLink>
        </ielement>
      </boundary>
    </actor>
  </diagram>
</istarml>
```

Example 7.2 Implementing “why“ as intentional relationship




```
<?xml version="1.0"?>
<istarml version="1.0">
  <diagram name="Example 7.2">
    <ielement id="21" type="goal" name="G2"/>
    <ielement name="G1" type="goal">
      <ielementLink iref="21" type="why"/>
    </ielement>
  </diagram>
</istarml>
```

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




```
1   <?xml version="1.0"?>
2   <istarml version="1.0">
3     <diagram name="Example 7.3">
4       <ielement id="123" name="T2" type="task"/>
5       <actor name="A">
6         <boundary>
7           <ielement id="125" name="T1" type="task">
8             <ielementLink type="decomposition" value="or">
9               <ielement iref="123"/>
10            </ielementLink>
11          </ielement>
12        </boundary>
13      </actor>
14      <ielement name="G" type="goal">
15        <ielementLink type="means-end">
16          <ielement iref="125"/>
17        </ielementLink>
18        <ielementLink type="decomposition" value="and">
19          <ielement iref="123"/>
20        </ielementLink>
21        <ielementLink type="STconstraint" value="restricts">
22          <ielement name="SF" type="softgoal"/>
23        </ielementLink>
24      </ielement>
25    </diagram>
26  </istarml>
```

8 Representing Dependencies

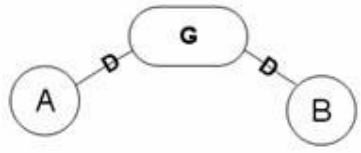
Dependencies is one of the classical *i**'s constructs and its aim is representing intentional relationships between two (or occasionally more) actors. To feature this relationship a specific intentional element makes the link among the involved actors which are named dependers or dependees. It represents 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.

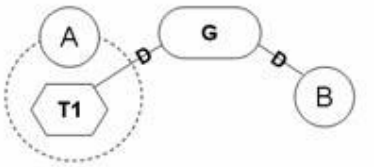
dependencyTag ::=	<dependency> *dependerTag* {*dependerTag*} {*dependeeTag*} </dependency >
dependerTag ::=	<depender [iref="*string*"] aref="*string*" [value="*dep-type*"] /> | <depender [iref=" *string*"] aref="*string*" [value="*dep-type*"] > [*graphic-path*] </depender>
dependeeTag ::=	<dependee [iref="*string*"] aref="*string*" [value="*dep-type*"] /> | <dependee [iref="*string*"] aref="*string*" [value="*dep-type*"] > [*graphic-path*] </dependee>
Dep-type ::=	open | committed | critical | delegation | permission | trust | owner | *string*

Example 8.1 Basic representation of dependency



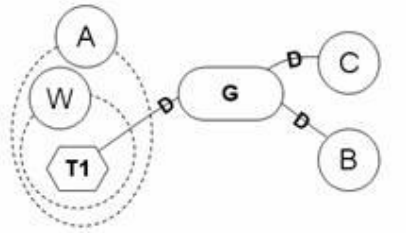

```
1   <?xml version="1.0"?>
2   <istarml version="1.0">
3     <diagram name="Example 8.1">
4       <actor id="21" name="A"/>
5       <actor id="10" name="B"/>
6       <ielement type="goal" name="G">
7         <dependency>
8           <depender aref="21"/>
9           <dependee aref="10"/>
10        </dependency>
11      </ielement>
12    </diagram>
13  </istarml>
```

Example 8.2 Dependency from an internal intentional element



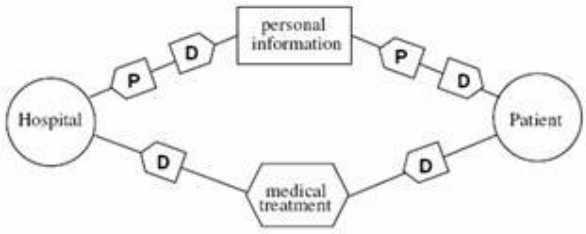

```
1   <?xml version="1.0"?>
2   <istarml version="1.0">
3     <diagram name="Example 8.2">
4       <actor id="201" name="A">
5         <boundary>
6           <ielement id="101" type="task" name="T1"/>
7         </boundary>
8       </actor>
9       <actor id="230" name="B"/>
10      <ielement name="G" type="goal">
11        <dependency>
12          <depender iref="101" aref="201"/>
13          <dependee aref="230"/>
14        </dependency>
15      </ielement>
16    </diagram>
17  </istarml>
```

Example 8.3 Dependency from a nested actor to multiple dependees



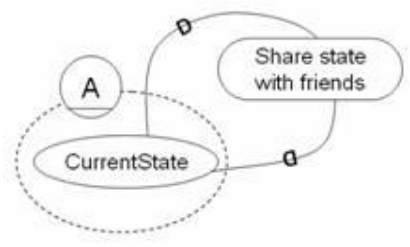

```
1   <?xml version="1.0"?>
2   <istarml version="1.0">
3     <diagram name="Example 8.3">
4       <actor id="201" name="A">
5         <boundary>
6           <actor id="20" name="W">
7             <boundary>
8               <ielement id="101" type="task" name="T1"/>
9             </boundary>
10          </actor>
11        </boundary>
12      </actor>
13      <actor id="230" name="B"/>
14      <actor id="231" name="C"/>
15      <ielement name="G" type="goal">
16        <dependency>
17          <depender iref="101" aref="20"/>
18          <dependee aref="230"/>
19          <dependee aref="231"/>
20        </dependency>
21      </ielement>
22    </diagram>
23  </istarml>
```

Example 8.4 Extended dependencies from Secure Tropos [10, 22]



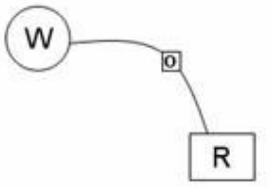
```
1  <?xml version="1.0"?>
2  <istarml version="1.0">
3    <diagram name="Example 8.4">
4      <actor id="X1" name="Hospital"/>
5      <actor id="A2" name="Patient"/>
6      <ielement type="resource" name="Personal information">
7        <dependency>
8          <depender aref="X1" value="delegation"/>
9          <depender aref="A2" value="permission"/>
10         <dependee aref="A2" value="delegation"/>
11         <dependee aref="X1" value="permission"/>
12       </dependency>
13     </ielement>
14     <ielement type="task" name="medical treatment">
15       <dependency>
16         <depender aref="A2" value="delegation"/>
17         <dependee aref="X1" value="delegation"/>
18       </dependency>
19     </ielement>
20   </diagram>
21 </istarml>
```

Example 8.5 Abstract self dependency taken from Tropos-PL [23]



```
1  <?xml version="1.0"?>
2  <istarml version="1.0">
3    <diagram name="Example 8.5">
4      <actor id="201" name="A" type="role">
5        <boundary>
6          <ielement id="101" type="belief" name="T1"/>
7        </boundary>
8      </actor>
9      <ielement name="Share state with friends" type="goal">
10       <dependency>
11         <depender iref="101" aref="201"/>
12         <dependee iref="101" aref="201"/>
13       </dependency>
14     </ielement>
15   </diagram>
16 </istarml>
```

Example 8.6 Representing the owner relationship from Secure Tropos [22]




```
1  <?xml version="1.0"?>
2  <istarml version="1.0">
3    <diagram name="Example 8.6">
4      <actor id="A21" name="W"/>
5      <ielement id="I22" name="R" type="resource">
6        <dependency>
7          <depender aref="A21" value="owner"/>
8        </dependency>
9      </ielement>
10   </diagram>
11 </istarml>
```

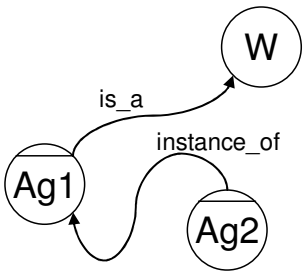
9 Representing actor's relationships

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

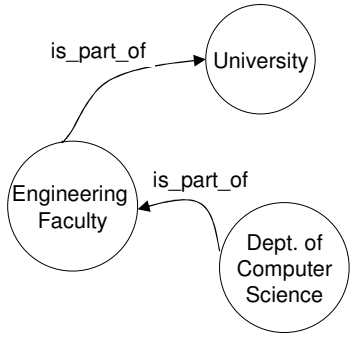
actorLinkTag ::=	<actorLink type="*actorLink-type*" aref="*string*"> [*graphic-path*] </actorLink> | <actorLink type="*actorLink-type*" aref="*string*"/>
actorLink-type ::=	is_part_of | is_a | instance_of | plays | covers | occupies | *string*

Example 9.1 Representing *instance_of* (INS) and *is_a* relationships



```
1  <?xml version="1.0"?>
2  <istarml version="1.0">
3    <diagram name="Example 9.1">
4      <actor id="32" name="W"/>
5      <actor id="202" name="Ag1" type="agent">
6        <actorLink type="is_a" aref="32"/>
7      </actor>
8      <actor name="Ag2" type="agent">
9        <actorLink type="instance" aref="202"/>
10     </actor>
11   </diagram>
12 </istarml>
```

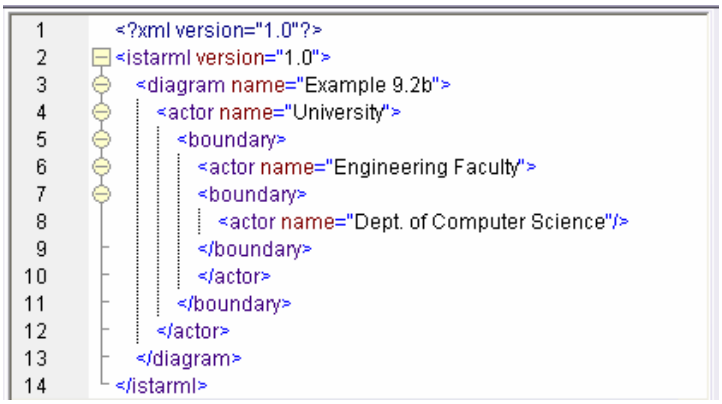
Example 9.2 The two representations for *is_part_of* relationship



a) Using <actorLink>

```
1  <?xml version="1.0"?>
2  <istarml version="1.0">
3    <diagram name="Example 9.2a">
4      <actor id="201" name="University"/>
5      <actor id="202" name="Engineering Faculty">
6        <actorLink type="is_part_of" aref="201"/>
7      </actor>
8      <actor id="203" name="Dept. of Computer Science">
9        <actorLink type="is_part_of" aref="202"/>
10     </actor>
11   </diagram>
12 </istarml>
```

b) Using nested structures



```
1   <?xml version="1.0"?>
2   <istarml version="1.0">
3     <diagram name="Example 9.2b">
4       <actor name="University">
5         <boundary>
6           <actor name="Engineering Faculty">
7           <boundary>
8             <actor name="Dept. of Computer Science"/>
9           </boundary>
10          </actor>
11        </boundary>
12      </actor>
13    </diagram>
14  </istarml>
```

10 iStarML's Graphic specification

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] .

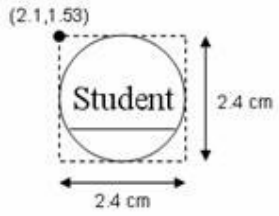
Therefore, we account with two alternative ways of specifying graphic expressions. Both are present in our EBNF specification showed at table 10.1.

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

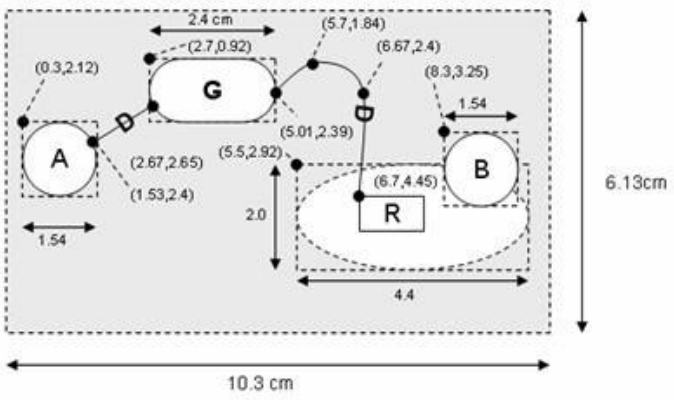
graphic-path ::=	<graphic content="SVG"> *svg-content* </graphic> |
	<graphic content="basic" *g-options-path*>
	<point xpos="*number*" ypos="*number*" />
	<point xpos="*number*" ypos="*number*" />
	{<point xpos="*number*" ypos="*number*"/>}
	</graphic> |
	<graphic content="basic" *g-options-shape*/>
g-options-shape ::=	xpos="*number*" "ypos="*number*"
	width="*number*" height="*number*"
	shape="*shape*" [unit="*unit*"]
	[bgcolor="*hexrgbcolor*"] [fontcolor="*hexrgbcolor*"]
	[fontfamily="*string*"] [fontsize="*number*"]
g-options-path ::=	shape="*irregular*" [unit="*unit*"]
	[bgcolor="*hexrgbcolor*"]
	[fontcolor="*hexrgbcolor*"]
	[fontfamily="*string*"] [fontsize="*number*"]
irregular ::=	polyline | spline
shape ::=	ellipse | rect

Example 10.1 Basic coordinates in graphic representations




```
1   <?xml version="1.0"?>
2   <istaml version="1.0">
3     <diagram name="Example 10.1">
4       <actor name="Student">
5         <graphic content="basic" xpos="2.1"
6                  ypos="1.53" unit="cm"
7                  width="2.4" height="2.4"/>
8       </actor>
9     </diagram>
10  </istaml>
```

Example 10.2 Combining graphic tags to represent a complete diagram




```
<?xml version="1.0"?>
<istarml version="1.0">
  <diagram name="Example 10.2">
    <graphic content="basic" width="10.3" height="6.13"
      unit="cm" bgcolor="aeaeae"/>
    <actor id="11" name="A">
      <graphic content="basic" xpos="0.3" ypos="2.12"
        width="1.54" height="1.54" unit="cm" fontfamily="arial" fontsize="14"/>
    </actor>
    <actor id="12" name="B">
      <graphic content="basic" xpos="8.3" ypos="3.25" width="1.54"
        height="1.54" unit="cm" fontfamily="arial" fontsize="14"/>
      <boundary>
        <graphic content="basic" xpos="5.5" ypos="2.92" shape="ellipse"
          width="4.4" height="2.0" unit="cm"/>
        <ielement id="7" type="resource" name="R">
          <graphic content="basic" xpos="6.7" ypos="4.45" unit="cm"
            width="1.54" height="0.78"/>
        </ielement>
      </boundary>
    </actor>
    <ielement name="G" type="goal">
      <graphic content="basic" xpos="2.7" ypos="0.92" width="2.4"
        height="1.7" unit="cm"/>
      <dependency>
        <depender aref="11">
          <graphic content="basic" shape="spline" unit="cm">
            <point xpos="1.53" ypos="2.4"/>
            <point xpos="2.67" ypos="2.65"/>
          </graphic>
        </depender>
        <dependee iref="7" aref="12">
          <graphic content="basic" shape="spline" unit="cm">
            <point xpos="5.01" ypos="2.39"/>
            <point xpos="5.7" ypos="1.84"/>
            <point xpos="6.67" ypos="2.4"/>
            <point xpos="6.7" ypos="4.45"/>
          </graphic>
        </dependee>
      </dependency>
    </ielement>
  </diagram>
</istarml>
```

The way to use 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">
                </svg>
            </graphic>
        .
        .
        .
```

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>

        .
        .
        .
```

Example 10.5 Intentional element with an SVG graphic representation

```
<ielement 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>
        <path fill="none" stroke="#3344FF" stroke-width="2"
           d="M130,100 C210,140 290,140 380,100 S450,350 370,300
            S210,260 120,300 S50,60 130,100"/>
        </g>
</graphic>
</ielement>
```

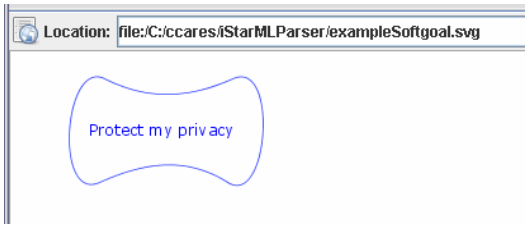
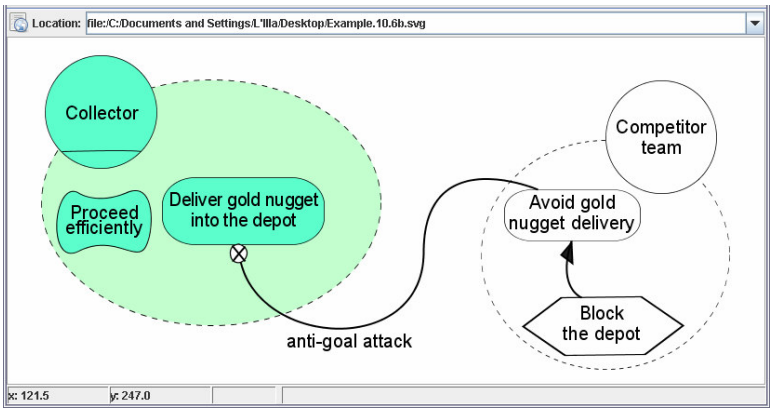


Figure 10.1 SVG display of the code portion from the example 10.5

Example 10.6 A portion of the diagram extracted from [26] and its iStarML code




<?xml version="1.0"?>
<istarml version="1.0">
<diagram name="Example 10.6">
<graphic content="SVG">
<svg xmlns="http://www.w3.org/2000/svg" width="210mm" height="297mm">
</svg>
</graphic>
<actor name="Collector" type="role">
<graphic content="SVG">
<boundary>
</actor>
<actor name="Competitor team">
<graphic content="SVG">
<boundary>
<graphic content="SVG">
<g>
<path style="fill:none;fill-opacity:1;stroke:#000000;stroke-width:0.5;stroke-r
4;stroke-dashoffset:0;stroke-opacity:1" id="path9052" d="M 323.31821 231.6
80.829559,231.63136 A 121.24432 70.785645 0 1 1 323.31821 231.63136
matrix(0.5509308,0,0,0.8693309,253.83909,-15.169662)"/>
</g>
</graphic>
<ielement id="agn01" type="goal" name="Avoid gold nugget delivery">
<graphic content="SVG">
<ielementLink type="means-end">
<graphic content="SVG">
<ielement iref="bdd01"/>
</ielementLink>
</ielement>
<ielement id="bdd01" type="task" name="Block the depot">
<graphic content="SVG">
</ielement>
</boundary>
</actor>
</diagram>
</istarml>

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 are restricted to the *actor* and *ielement* tags.

Appendix A. Complete code of example 10.6

```
1   <?xml version="1.0"?>
2   <istarml version="1.0">
3     <diagram name="Example 10.6">
4       <graphic content="SVG">
5         <svg xmlns="http://www.w3.org/2000/svg" width="210mm" height="297mm">
6         </svg>
7       </graphic>
8       <actor name="Collector" type="role">
9         <graphic content="SVG">
10          <g>
11            <path style="
    fill:#5cffcc;fill-opacity:1;fill-rule:evenodd;stroke:#000000;stroke-width:1px;stroke-linecap:
    butt;stroke-linejoin:miter;stroke-opacity:1;opacity:1" id="path2160" d="M 294.28572
    340.93362 A 81.428574 77.14286 0 1 1  131.42857,340.93362 A 81.428574
    77.14286 0 1 1  294.28572 340.93362 z" transform="
    matrix(0.3670502,0,0,0.3941223,15.665024,-24.840363)"/>
12            <path style="
    fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:0.44294775px;stroke-linecap:but
    t;stroke-linejoin:miter;stroke-opacity:1" d="M 72.877985,130.74476 C
    113.6252,129.69708 115.00646,130.74476 115.00646,130.74476"/>
13            <text style="
    font-size:10px;font-style:normal;font-variant:normal;font-weight:normal;font-stretch:norm
    al;text-align:center;line-height:100%;writing-mode:lr-tb;text-anchor:middle;fill:#000000;fi
    ll-opacity:1;stroke:none;stroke-width:1px;stroke-linecap:butt;stroke-linejoin:miter;stroke-
    opacity:1;font-family:Arial" x="94.098068" y="113.68563" id="text3149">
14              <tspan id="tspan3151" x="94.098068" y="113.68563">Collector</tspan>
15            </text>
16          </g>
17        </graphic>
18        <boundary>
19          <graphic content="SVG">
20            <g>
21              <path style="
    fill:#c3ffce;fill-opacity:1;stroke:#000000;stroke-width:0.5;stroke-miterlimit:4;stroke-dash
    array:4, 4;stroke-dashoffset:0;stroke-opacity:1" id="path3165" d="M 323.31821
    231.63136 A 121.24432 70.785645 0 1 1  80.829559,231.63136 A 121.24432
```

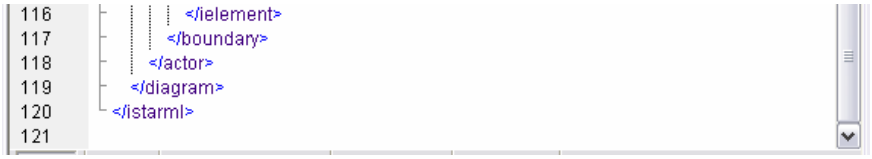
```
70.785645 0 1 1  323.31821 231.63136 z" transform="
matrix(0.7537876,0,0,0.9296294,0.7292643,-57.116145)"/>
22          </g>
23        </graphic>
24        <ielement type="softgoal" name="Proceed Efficiently">
25          <graphic content="SVG">
26            <g>
27              <path style="
fill:#5cffcc;fill-opacity:1;fill-rule:evenodd;stroke:#000000;stroke-width:0.48336101px;stro
ke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1" d="M 81.335123,152.53531 C
81.335123,152.53531 90.969961,159.6023 107.68346,152.53531 C
124.39696,145.4683 121.84078,177.54161 116.53178,183.24957 C
111.22279,188.95753 99.031771,175.36715 81.531753,184.3368 C
64.031733,193.30645 67.964323,146.01192 81.335123,152.53531 z " id="path8065"/>
28              <text style="
font-size:10px;font-style:normal;font-variant:normal;font-weight:normal;font-stretch:norm
al;text-align:center;line-height:80.00000119%;writing-mode:lr-tb;text-anchor:middle;fill:#
000000;fill-opacity:1;stroke:none;stroke-width:1px;stroke-linecap:butt;stroke-linejoin:mit
er;stroke-opacity:1;font-family:Arial" x="95.178001" y="167.06335" id="text9036">
29                <tspan id="tspan9038" x="95.178001" y="167.06335">Proceed</
tspan>
30                <tspan x="95.178001" y="175.06335" id="tspan9040">efficiently</
tspan>
31              </text>
32            </g>
33          </graphic>
34        </ielement>
35        <ielement type="goal" name="Deliver gold nugget into the depot">
36          <graphic content="SVG">
37            <g>
38              <a id="a9042" transform="
matrix(1.3757789,0,0,1.1339694,-407.41334,46.882428)">
39                <rect ry="12.634748" y="86.291222" x="388.21475" height="
31.790661" width="63.358788" id="rect9044" style="
fill:#5cffcc;fill-opacity:1;fill-rule:evenodd;stroke:#000000;stroke-width:0.28664446px;stro
ke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1"/>
```


40
41 <text style="
font-size:9.54275227px;font-style:normal;font-variant:normal;font-weight:normal;font-stretch:normal;text-align:center;line-height:110.00000238%;writing-mode:lr-tb;text-anchor:middle;fill:#000000;fill-opacity:1;stroke:none;stroke-width:1px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1;font-family:Arial" x="174.21478" y="156.59018" id="text9046" transform="scale(0.977482,1.0230368)">
42 <tspan id="tspan9048" x="174.21478" y="156.59018">Deliver gold nugget</tspan>
43 <tspan x="174.21478" y="167.08721" id="tspan9050">into the depot</tspan>
44 </text>
45 </g>
46 </graphic>
47 <ielementLink type="anti-goal">
48 <graphic content="SVG">
49 <g>
50 <text style="
font-size:10px;font-style:normal;font-variant:normal;font-weight:normal;font-stretch:normal;text-align:center;line-height:100%;writing-mode:lr-tb;text-anchor:middle;fill:#000000;fill-opacity:1;stroke:none;stroke-width:1px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1;font-family:Arial" x="227.18365" y="235.9359" id="text4537">
51 <tspan id="tspan4539" x="227.18365" y="235.9359">anti-goal attack</tspan>
52 </text>
53 <path style="
fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:1px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1" d="M 328.5793,151.28009 C 328.5793,151.28009 267.35928,124.97462 267.35928,184.75979 C 267.35928,244.54496 167.87676,235.9359 167.87676,180.45526" id="path2575"/>
54 <path style="
opacity:1;fill:#ffffff;fill-opacity:1;stroke:#000000;stroke-width:0.60000002;stroke-miterlimit:4;stroke-dasharray:none;stroke-dashoffset:0;stroke-opacity:1" id="path2577" d="M 111.91784 254.828 A 4.543673 5.0219545 0 1 1 102.8305,254.828 A 4.543673 5.0219545 0 1 1 111.91784 254.828 z" transform="translate(60.263452,-69.350798)" />

```
55                  <path style="
    fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:1px;stroke-linecap:butt;stroke-lin
    ejoin:miter;stroke-opacity:1" d="M 164.05051,181.41182 L 171.22473,189.5426" id="
    path2579"/>
56                  <path style="
    fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:1px;stroke-linecap:butt;stroke-lin
    ejoin:miter;stroke-opacity:1" d="M 171.22473,181.8901 C 164.52879,189.5426
    164.52879,189.5426 164.52879,189.5426" id="path2581"/>
57                </g>
58              </graphic>
59              <ielement iref="agn01"/>
60              <!-- to test without this line -->
61            </ielementLink>
62          </ielement>
63        </boundary>
64      </actor>
65      <actor name="Competitor team">
66        <graphic content="SVG">
67          <g>
68            <path style="
    opacity:1;fill:#ffffff;fill-opacity:1;fill-rule:evenodd;stroke:#000000;stroke-width:1px;stroke-l
    inecap:butt;stroke-linejoin:miter;stroke-opacity:1" id="path9072" d="M 294.28572
    340.93362 A 81.428574 77.14286 0 1 1  131.42857,340.93362 A 81.428574
    77.14286 0 1 1  294.28572 340.93362 z" transform="
    matrix(0.3670502,0,0,0.3941223,316.93115,-11.308051)"/>
69            <text style="
    font-size:10px;font-style:normal;font-variant:normal;font-weight:normal;font-stretch:norm
    al;text-align:center;line-height:110.00000238%;writing-mode:lr-tb;text-anchor:middle;fill:
    #000000;fill-opacity:1;stroke:none;stroke-width:1px;stroke-linecap:butt;stroke-linejoin:m
    iter;stroke-opacity:1;font-family:Arial" x="395.06039" y="121.14837" id="text4525">
70              <tspan id="tspan4527" x="395.06039" y="121.14837">Competitor</
    tspan>
71              <tspan x="395.06039" y="132.14837" id="tspan4529">team</tspan>
72            </text>
73          </g>
74        </graphic>
```


75 <boundary>
76 <graphic content="SVG">
77 <g>
78 <path style="
fill:none;fill-opacity:1;stroke:#000000;stroke-width:0.5;stroke-miterlimit:4;stroke-dasharray:4,
79 4;stroke-dashoffset:0;stroke-opacity:1" id="path9052" d="M 323.31821 231.63136 A 121.24432 70.785645 0 1 1
80 80.829559,231.63136 A 121.24432 70.785645 0 1 1 323.31821 231.63136 z" transform="
81 matrix(0.5509308,0,0,0.8693309,253.83909,-15.169662)"/>
82 </g>
83 </graphic>
84 <ielement id="agn01" type="goal" name="Avoid gold nugget delivery">
85 <graphic content="SVG">
86 <g>
87 <a id="a10043" transform="
matrix(1.1502419,0,0,0.8658276,-196.96761,166.5912)">
88 <rect ry="12.634748" y="-17.559587" x="441.43835" height="31.790661" width="63.358788" id="rect10045" style="
fill:none;fill-opacity:1;fill-rule:evenodd;stroke:#000000;stroke-width:0.28664446px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1"/>
89
90 <text style="
font-size:10px;font-style:normal;font-variant:normal;font-weight:normal;font-stretch:normal;text-align:center;line-height:110.00000238%;writing-mode:lr-tb;text-anchor:middle;fill:#000000;fill-opacity:1;stroke:none;stroke-width:1px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1;font-family:Arial" x="346.754" y="161.32399" id="text10047">
91 <tspan id="tspan10049" x="346.754" y="161.32399">Avoid gold</tspan>
92 <tspan x="346.754" y="172.32399" id="tspan10051">nugget delivery</tspan>
93 </text>
94 </g>
95 </graphic>
96 <ielementLink type="means-end">

```
97          <graphic content="SVG">
98            <g>
99              <path style="
fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:1px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1" d="M 352.49336,209.15214 C 352.49336,209.15214 338.6232,203.89104 346.75399,178.54213" id="path2583"/>
100             <path style="
fill:#000000;fill-opacity:0.877095;fill-rule:evenodd;stroke:#000000;stroke-width:1px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1" d="M 341.315,186.41664 L 346.5761,179.7207 L 347.05438,190.72118 L 341.315,186.41664 z" id="path2585"/>
101           </g>
102         </graphic>
103         <ielement iref="bdd01"/>
104       </ielementLink>
105     </ielement>
106     <ielement id="bdd01" type="task" name="Block the depot">
107       <graphic content="SVG">
108         <g>
109           <path style="
opacity:1;fill:#ffffff;fill-opacity:1;stroke:#000000;stroke-width:0.60000002;stroke-miterlimit:4;stroke-dasharray:none;stroke-dashoffset:0;stroke-opacity:1" id="path2573" d="M 42.088758,238.32731 L 32.724565,222.97999 L 41.33363,207.19671 L 59.306888,206.76074 L 68.67108,222.10805 L 60.062015,237.89133 L 42.088758,238.32731 z" transform="
matrix(2.3880924,0,0,0.9897848,242.66188,4.1864602)"/>
110           <text style="
font-size:10px;font-style:normal;font-variant:normal;font-weight:normal;font-stretch:normal;text-align:center;line-height:110.00000238%;writing-mode:lr-tb;text-anchor:middle;fill:#000000;fill-opacity:1;stroke:none;stroke-width:1px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1;font-family:Arial" x="363.01556" y="221.10918" id="text4531">
111             <tspan id="tspan4533" x="363.01556" y="221.10918">Block</tspan>
112             <tspan x="363.01556" y="232.10918" id="tspan4535">the depot</
tspan>
113           </text>
114         </g>
115       </graphic>
```


116 </ielement>
117 </boundary>
118 </actor>
119 </diagram>
120 </istarml>
121

References

[1] D. C. Fallside and P. Walmsley, "XML Schema Part 0: Primer", http://www.w3.org/TR/xmlschema-0/ last accessed (2004)

[2] E. Yu, "Modelling Strategic Relationships for Process Reengineering", in *Computer Science*, vol. PhD. Toronto: University of Toronto, 1995.

[3] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology", *Autonomous Agents And Multi-Agent Systems*, vol. 8, pp. 203-236, 2004.

[4] J. Castro, M. Kolp, and J. Mylopoulos, "A Requirements-Driven Development Methodology", *Advanced Information Systems Engineering: 13th International Conference, CAiSE 2001,Interlaken, Switzerland*, pp. 108-123, 2001.

[5] L. Cernuzzi, M. Cossentino, and F. Zambonelli, "Process models for agent-based development", *Engineering Applications of Artificial Intelligence*, vol. 18, pp. 205–222, 2005.

[6] K. H. Dam and M. Winikoff, "Comparing agent-oriented methodologies", in *Agent-Oriented Information Systems*, vol. 3030, *Lecture Notes In Computer Science*. Berlin: Springer-Verlag Berlin, 2003, pp. 78-93.

[7] J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf, "Evaluation of Agent–Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform", *AOSE 2004. Lecture Notes in Computer Science*, vol. 3382, pp. 126-141, 2005.

[8] "GRL - Goal Oriented Requirement Language", http://www.cs.toronto.edu/km/GRL/ last accessed September 20, 2005 (n.d.)

[9] "Z.150: User Requirements Notation (URN) - Language requirements and framework ", http://www.itu.int/rec/ recommendation.asp?type =folders&lang =e&parent= T-REC-Z.150 last accessed Sept)

[10] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Requirements Engineering Meets Trust Management, Model, Methodology, and Reasoning", *Lecture Notes in Computer Science*, vol. 2995, pp. 176-190, 2004.

[11] L. Liu, E. Yu, and J. Mylopoulos, "Security and Privacy Requirements Analysis within a Social Setting", presented at International Conference on Requirements Engineering (RE'03), Monterey, California, USA, 2003.

[12] H. Mouratidis, M. Weiss, and P. Giorgini, "Security Patterns Meet Agent Oriented Software Engineering: A Complementary Solution for Developing Secure Information Systems ", in *Conceptual Modeling – ER 2005: 24th International Conference on Conceptual Modeling*, vol. 3716, *Lecture Notes in Computer Science*, L. Delcambre, C. Kop, H. C. Mayr, J. Mylopoulos, and O. Pastor, Eds.: Springer Verlag, 2005, pp. 225 - 240.

[13] A. Fuxman, L. Liu, J. Mylopoulos, and M. Pistore, "Specifying and analyzing early requirements in Tropos", *Requirements Engineering*, vol. 9, pp. 132-150, 2004.

[14] G. Grau, X. Franch, E. Mayol, C. Ayala, C. Cares, J. P. Carvallo, M. Haya, F. J. Navarrete, P. Botella, and C. Quer, "RiSD: A Methodology for Building i* Strategic Dependency Models", presented at Proc. of the 17th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'05), Taipei, Taiwan; China, 2005.

[15] E. Yu, "Modeling organizations for information systems requirements engineering", *Proceedings of IEEE International Symposium on Requirements Engineering. San Diego, CA, USA*, pp. 34-41, 1993.

[16] G. Grau, C. Cares, X. Franch, and F. Navarrete, "A Comparative Analysis of i* Agent-Oriented Modelling Techniques", presented at 18th International Conference on Software Engineering and Knowledge Engineering (SEKE'06), San Francisco, California, USA, 2006.

[17] M. Kolp, P. Giorgini, and J. Mylopoulos, "Organizational Patterns for Early Requirements Analysis ", *Lecture Notes in Computer Science*, vol. 2681, pp. 617-632, 2003.

[18] P. Donzelli, "A goal-driven and agent-based requirements engineering framework", *Requirements Engineering*, vol. 9, 2004.

[19] R. Sethi, *Programming Languages: Concepts and Constructs (2nd edition).* Addison Wesley, 1996.

[20] P. Donzelli and R. Setola, "Handling the knowledge acquired during the requirements engineering process - a case study -", presented at Proceedings of the 14th Software Engineering and Knowledge Engineering Conference, SEKE'02, Ischia, Italy, 2002.

[21] F. Sannicolo, A. Perini, and F. Giunchiglia, "The Tropos Modeling Language. A User Guide", Informatica e Telecomunicazioni, University of Trento., Technical Report DIT-02-061, 2002.

[22] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Modeling Social and Individual Trust in Requirements Engineering Methodologies", *Lecture Notes in Computer Science*, vol. 3477, pp. 161–176, 2005.

[23] C. Cares, X. Franch, and E. Mayol, "Extending Tropos for a Prolog Implementation: A Case Study Using the Food Collecting Agent Problem (Award)", presented at Sixth International Workshop Computational Logic in Multi-Agent Systems, London, UK, 2005.

[24] "Scalable Vectors Graphics, SVG", http://www.w3.org/TR/SVG11/ last accessed July 1, 2007 (2003)

[25] "SVGI, Scalable Vector Graphics Implementations", http://www.svgi.org/ last accessed July 1, 2007

[26] C. Cares, X. Franch, and E. Mayol, "Using Antimodels to Define Agents' Strategy", *Lecture Notes in Computer Science*, vol. 4371, pp. 284-293, 2006.