Title: Graphic GRAFCET diagram editor based on GeCé
Student: Samuel Gómez Crespo
Supervisor: José Pedro Muñoz Gázquez
Tutor: Antonio Benito Martínez Velasco
Department: Service and Information System Engineering
Presentation date: 2011-09-21
PROJECT DATA

Project title: Graphic GRAFCET diagram editor based on GeCè

Student name: Samuel Gómez Crespo

Degree: Master's Degree in Information Technology specialized in Software Engineering and Information Systems

Credits: 30 ECTS

Supervisor: José Pedro Muñoz Gázquez

Tutor: Antonio Benito Martínez Velasco

Department: Service and Information System Engineering

BOARD MEMBERS (name and signature)

President: Xavier Franch Gutiérrez

Secretary: Cristina Gómez Seoane

Member: Jose M. Cabré Garcia

GRADE

Numeric grade:

Descriptive grade:

Date:
# Table of Contents

Introduction.......................................................................................................................... 13  
What is GRAFCET?............................................................................................................... 13  
  GRAFCET applications ...................................................................................................... 13  
What is CITCEA? .................................................................................................................. 14  
What is GeCé? ....................................................................................................................... 15  
Why translation to C? ........................................................................................................... 15  
  Why not C directly? ............................................................................................................ 15  
Project motivation .................................................................................................................. 15  
  A local problem .................................................................................................................. 15  
  A growing global problem ................................................................................................. 15  
  Introducing a solution ....................................................................................................... 15  
The goals of the project ......................................................................................................... 16  
Methodology .......................................................................................................................... 17  
  Initial methodology .......................................................................................................... 17  
  UniSim discovery .............................................................................................................. 17  
  Methodology revision ........................................................................................................ 17  
  Discarded methodology .................................................................................................... 18  
  Discovery of differences between SFC and GRAFCET ..................................................... 19  
  The ultimate methodology ............................................................................................... 19  
    Evolution of the design tasks ......................................................................................... 20  
    Modification from sprint 10 ......................................................................................... 21  
Business modeling ............................................................................................................... 22  
  Use case diagram ............................................................................................................ 22  
  Class diagram .................................................................................................................. 23  
    GRAFCET ....................................................................................................................... 23  
Requirements specification ...................................................................................................... 26  
  The need for good requirements ....................................................................................... 26  
Requirements composition .................................................................................................... 26  
Stakeholders .......................................................................................................................... 26  
  Current users of GeCé ....................................................................................................... 26  
  Other GRAFCET designers ............................................................................................. 26  
  Product owner ................................................................................................................... 26  
The scope of the work ........................................................................................................... 27  
  The current situation ........................................................................................................ 27  
  The new situation ............................................................................................................ 27  
  The context of the work ................................................................................................... 27  
Off-the-Shelf solutions ......................................................................................................... 27  
Functional requirements ....................................................................................................... 30  
  Product use case list ........................................................................................................ 30  
  Individual product use cases .......................................................................................... 30  
Conceptual schema ............................................................................................................... 41  
  Class diagram .................................................................................................................. 41  
  Changes over the business class diagram ......................................................................... 43  
Non-functional requirements ................................................................................................. 45  
  Usability and humanity requirements ........................................................................... 45  
  Maintainability and support requirements ..................................................................... 45  
Legal requirements ............................................................................................................... 45
Satisfaction argument................................................................. 45
Planning................................................................................... 47
Minor sprints........................................................................... 51
Features by sprint...................................................................... 52
Analysis................................................................................... 53
Code generation....................................................................... 53
  Macro-step code generation................................................ 53
Verification.............................................................................. 57
Design.................................................................................... 61
Design goals............................................................................ 61
Reusing UniSim....................................................................... 61
  Finding the reusable components........................................ 61
  Design model of the reusable components.......................... 64
  GUI design model verification............................................. 66
  Cross-platform design........................................................ 67
Code generation....................................................................... 69
  Actions.................................................................................. 69
Verification.............................................................................. 72
  Steps status......................................................................... 72
  Variables value..................................................................... 73
Adding new data types............................................................. 77
Design model.......................................................................... 78
  New project......................................................................... 78
  Save project......................................................................... 79
  Open project......................................................................... 80
  New grafcet.......................................................................... 82
  Add initial step...................................................................... 83
  Add step................................................................................ 85
  Add project variable............................................................. 86
  Add grafcet variable.............................................................. 87
  Add transition....................................................................... 88
Code generation....................................................................... 90
Run........................................................................................ 91
  Add action............................................................................ 96
  Add macro-step.................................................................... 98
  Add enclosing Step.............................................................. 101
  Add forcing order.............................................................. 106
Implementation........................................................................ 109
DRY improvements.................................................................. 109
  Snap..................................................................................... 109
  Small rectangles................................................................. 109
Implementation tradeoffs......................................................... 110
  Action graphics................................................................... 110
  Actions code generation................................................... 112
  CRUDs................................................................................... 112
Economic assessment.............................................................. 113
Material resources cost............................................................ 113
  Hardware............................................................................. 113
  Software............................................................................... 113
  Buildings............................................................................. 113
Sprint 7.1.........................................................................................................................135
Sprint planning.................................................................................................................135
Sprint review......................................................................................................................135
Sprint 8...............................................................................................................................136
Sprint planning.................................................................................................................136
Sprint review......................................................................................................................136
Sprint 9...............................................................................................................................138
Sprint planning.................................................................................................................138
Sprint review......................................................................................................................138
Sprint 10.............................................................................................................................140
Sprint planning.................................................................................................................140
Sprint review......................................................................................................................140
Sprint 10.1.........................................................................................................................141
Sprint planning.................................................................................................................141
Sprint review......................................................................................................................141
Sprint 10.2.............................................................................................................................142
Sprint planning.................................................................................................................142
Sprint review......................................................................................................................142
Sprint 11.............................................................................................................................143
Sprint planning.................................................................................................................143
Extraordinary meeting.................................................................................................143
Sprint review......................................................................................................................144
Sprint 12.............................................................................................................................145
Sprint planning.................................................................................................................145
Sprint review......................................................................................................................145
Sprint 13.............................................................................................................................146
Sprint planning.................................................................................................................146
Sprint review......................................................................................................................146
Sprint 14.............................................................................................................................147
Sprint planning.................................................................................................................147
Sprint review......................................................................................................................147
Sprint 15.............................................................................................................................148
Sprint planning.................................................................................................................148
Sprint review......................................................................................................................148
Sprint 16.............................................................................................................................149
Sprint planning.................................................................................................................149
Sprint review......................................................................................................................149
Sprint 17.............................................................................................................................150
Sprint planning.................................................................................................................150
Sprint review......................................................................................................................150
Future sprints....................................................................................................................151
Illustration Index

Illustration 1: Example of GRAFCET diagram .................................................................7
Illustration 2: Summary of the goals of the project ..........................................................13
Illustration 3: Example of features associated to more than one goal ..............................17
Illustration 4: Business use case diagram .....................................................................23
Illustration 5: Simplified business class diagram ............................................................25
Illustration 6: Complete business class diagram .............................................................27
Illustration 7: Screenshot of UniSim ...............................................................................35
Illustration 8: Screenshot of GRAFTOR (although the title reads “Grafedit”) ...............35
Illustration 9: System use case diagram .......................................................................37
Illustration 10: System class diagram. Part 1 .................................................................59
Illustration 11: System class diagram. Part 2 .................................................................61
Illustration 12: Planification at the beginning of the project ...........................................70
Illustration 13: Planification at the end of the project .....................................................72
Illustration 14: Sprint feedback as seen at the beginning (above) and at the end (below) of the project ..................................................................................................................73
Illustration 15: Features by sprint ....................................................................................74
Illustration 16: Excerpt from Committee Draft IEC 60848 Ed. 2: Expansion of the macro-step ..............................75
Illustration 17: Mapping between steps of the diagram and steps of the C code............77
Illustration 18: Wrong mapping between steps of the diagram and steps of the C code..77
Illustration 19: Correct mapping between steps of the diagram and steps of the C code. 79
Illustration 20: Detail of steps and transitions, from the conceptual schema class diagram 79
Illustration 21: Example of macro-step and its expansion .............................................81
Illustration 22: Input and output of grafcet verification actions .....................................85
Illustration 23: Initial assignation of responsibilities for grafcet verification ................87
Illustration 24: UniSim components ..............................................................................89
Illustration 25: Domain model of IEC 1131, containing SFC (source: DISANZO) ...........93
Illustration 26: Domain model of GRAFCET (source: self elaboration) .......................95
Illustration 27: SFC editor GUI working out of UniSim .................................................97
Illustration 28: Mono project logo .................................................................................98
Illustration 29: Dynamics of the image table ................................................................100
Illustration 30: Conceptual view of data flow of action translation ...............................100
Illustration 31: Conceptual view of data flow of action translation in two steps ............101
Illustration 32: Data flow of a generic lexer ..................................................................101
Illustration 33: Data flow of a generic lexer generator ...................................................101
Illustration 34: Technical view of data flow of the lexer generator used for the product ..102
Illustration 35: Technical data flow of action translation .............................................102
Illustration 36: UniSim's small rectangles for steps .......................................................137
Illustration 37: Small rectangles for steps of the new system ......................................137
Illustration 38: IEC60848 symbol for continuous action .............................................139
Illustration 39: IEC60848 symbol for time-dependent continuous action ....................139
Illustration 40: IEC60848 symbol for stored action on rising edge ...............................139
Illustration 41: IEC60848 different representations of the association of several actions at one step. ..............................................................................................................................141
Illustration 42: Excerpt from IEC60848 ed2.0 annex B3 .............................................141
Illustration 43: Several actions per step, as represented by UniSim ..............................141
Illustration 44: Several actions per step, as represented by the product .......................143
Illustration 45: Project menu of the product, before creating a project .........................155
Illustration 46: Project menu of the product, after creating a project

Illustration 47: Diagram edition interface of the product

Illustration 48: Run command in the project menu

Illustration 49: Diagram validation interface of the product

Illustration 50: Code generation option in the project menu

Illustration 51: Source files generated by the product

Illustration 52: Microchip MPLAB IDE

Illustration 53: Garage door while being opened

Illustration 54: Garage door control circuitry
Index of Tables

Table 1: Goals satisfied by available off-the-shelf solutions ................................................................. 35
Table 2: References for off-the-shelf solutions .......................................................................................... 35
Table 3: UniSim vs GRAFTOR .................................................................................................................. 35
Table 4: Satisfaction argument .................................................................................................................. 69
Table 5: Domain knowledge needed for the project .................................................................................. 71
Table 6: Automatic serialization and deserialization capabilities by .NET Framework version ............. 100
Table 7: Mono language support by .NET Framework version ................................................................. 101
Table 8: Tasks by verification transaction .................................................................................................. 107
Table 9: Techniques initially considered for verification tasks ................................................................. 107
Table 10: Adopted techniques by verification task .................................................................................. 108
Table 11: Excerpt from IEC61131-3 Table 10 – Elementary data types ................................................. 109
Drawing Index

Drawing 1: Typical application of GRAFCET.................................................................11
Drawing 2: Phases of the project..................................................................................21
Drawing 3: Technology stack of UniSim........................................................................100
Drawing 4: Current technology stack of the product....................................................101
Drawing 5: A possible technology upgrade for the product............................................101
Drawing 6: Basic verification communication diagram................................................105
Drawing 7: Roles for verification communications.......................................................105
Drawing 8: Communications for steps status retrieval................................................105
Drawing 9: Communications for variables value setting.............................................106
Drawing 10: Communications for variable value retrieval...........................................106
Introduction

What is GRAFCET?
The term GRAFCET is the acronym of “GRAPHe de Commande Etape-Transition”. It is a specification language for the functional description of the behavior of the sequential part of a control system. [IEC60848].

GRAFCET is defined in the standard IEC60848.

Briefly, a GRAFCET is explained by the following static and dynamic views:

- **Static view**
  - A GRAFCET is a sequence of steps, separated by transitions.
  - Several actions can be associated to each step.
  - Each transition has a condition.

- **Dynamic view**
  - At the beginning, only the initial step is active.
  - When the condition of a transition is assessed as true, and its immediate previous steps are active, the immediate previous steps become inactive, and the immediate following steps become active.
  - Only actions of active steps are executed.

**GRAFCET applications**
Examples of control systems governable by GRAFCET are found in the following industrial sectors.

- **Automotive**
  - Assembly lines, soldering, painting.
- Tool machines: Lathes, milling machines, drills.
- Chemical and petrochemical plants
  - Process control: Dosage, mixing, weight-in.
  - Electrolytic baths, pipelines, refinement, water purification.
- Domotics
  - Lighting, ambiance temperature, anti-theft systems.
- Energy transformation
  - Electric plants, turbines, solar energy.
- Food
  - Packing, bottling, storage.
- Metallurgy
  - Oven control, lamination, smelting, soldering, forge, cranes.
- Paper and wood
  - Process control, chain saws, conglomerate and laminate production.
- Traffic
  - Control and regulation of traffic, railways, metro lines.

![Diagram](image)

In general, GRAFCETs are executed by automatons, which are the brain of the control system of automated sequential processes.

**What is CITCEA?**

“CITCEA” is the acronym for “Centre d'Innovació Tecnològica en Tecnologia de Convertidors Estàtics i Accionaments”, which means Centre for Technological Innovation in Static Converters and Actuators.

CITCEA is a research body of the Centre de Transferència de Tecnologia (CTT) of the Universitat Politècnica de Catalunya (UPC), and member of TECNIO.

CITCEA mission is to develop and transfer electrical technology mainly for companies in multiple industrial sectors.
CITCEA has expertise in mechatronics and enertronics fields, with a special incidence in power electronics and digital control.

One activity frequently performed at CITCEA is designing GRAFCET diagrams that will be executed by microcontrollers.

**What is GeCé?**

GeCé is a guide for compiling manually from GRAFCET language defined in the standard IEC 60848 ed2.0 to C language. [BAYO]

GeCé was developed at CITCEA on the year 2005.

**Why translation to C?**

C is the programming language with which many microcontrollers in the market are programmed. [BAYO]

**Why not C directly?**

C code is powerful for program and algorithm implementation, but clearly insufficient for specification, definition, and maintenance of a program. [BAYO]

**Project motivation**

**A local problem**

Stakeholders needed a quick and accessible way to edit, verify, and translate to C their GRAFCET diagrams.

**A growing global problem**

While in 2001, the software cost seldom exceeded the 10% of the cost of an automated system [BONFATTI], in 2011 this ratio has grown up to the highest part of it [PLCOPEN] [SUDRIA].

To predict whether this tendency is going to be kept or is going to shrink, we can observe three factors that have had a significant influence to it:

1. Increasing hardware standardization has contributed to lower hardware costs.
2. Increasing programming capabilities of industrial microprocessors has allowed for more flexible control units, ready to manage more complexity by employing more complex software.
3. It seems to exist an unlimited global demand for making things more efficiently.

As it seems that these factors are going to stay with us during the coming years, we can say that the need for solutions to this problem is going to grow.

**Introducing a solution**

This project aims to provide a GRAFCET editor and verificator, capable of translating the diagrams to C, by employing the methodology described in GeCé.
The goals of the project

The main goal of the project is to create a system that allows those interested in GRAFCET diagrams (grafcets), such as industrial programmers, industrial engineers, owners of automated systems, teachers, and students:

• To create, edit, and visualize grafcets without the need for additional training.
  ◦ Assuming they already have basic knowledge about GRAFCET structure and behavior.
• To verify the design of the grafct.
• To exchange grafcets with other users in an interoperable format.
• To translate these grafcets to C following the GeCé methodology.
• All together by focusing on the intrinsic value of the grafcets, and by benefiting from a graphical representation with which they are familiarized; without the need for analyzing, writing, reading or interpreting C code.

The definition of GRAFCET that stakeholders are interested in, is that provided by the standard IEC 60848 ed2.0.

Illustration 2: Summary of the goals of the project.
Methodology

Initial methodology
The initial order was to develop an editor from scratch.
Based on Volere's Requirements Specification Template [VOLERE], the following methodology was considered:

1. Requirements engineering
   1. Identifying stakeholder goals
   2. Designing requirements that meet that goals
2. Specification of the new system
3. Analysis, Design, Implementation, Testing

Candidate technologies for the implementation were Python [PYTHON] and GTK+ [GTK] due to their ease and interoperability.

UniSim discovery
Short after the start of the requirements engineering tasks, the UniSim [UNISIM] editor was found. UniSim is an editor for IEC 1131-3. This standard defines several programming languages for sequential processes. One of this languages is Sequential Function Chart (SFC).
At the moment of finding, the interviewed stakeholders and the read documentation pointed out that SFC was equivalent to GRACFCET, so UniSim was then initially assessed as a software to build this project upon.
Among the features of UniSim, obvious and remarkable were:

1. It had a working SFC-standard diagram editor
2. It had a working simulator
3. It had a working SFC-standard export feature
4. It was based on the work of four Italian B. Sc. thesis from years 2005 to 2007
5. It was licensed under GPL [GPL]
6. It was written in Visual Basic 2005 [VB]
7. Its documentation was in Italian

Although points 6 and 7 implied some development and deployment constraints, UniSim passed that initial assessment.

Methodology revision
Due to the discovery of UniSim, the methodology was reconsidered, to have these phases:

1. Requirements engineering
   1. Identifying stakeholder goals
   2. Identifying which goals are already met by which features of UniSim.
3. Identifying which features don’t serve to any goal and consider its removal.
4. Thinking new features or modification of existing features of UniSim that meet the unmet goals.
2. Specification of modifications/additions/removals over UniSim
3. Analysis, Design, Implementation, Testing

As it can be seen, this methodology prioritized code reuse over resemblance to an ideal solution.

**Discarded methodology**

Regarding the methodology for engineering of new requirements, the following one was discarded:

1. Identifying stakeholders goals.
2. Thinking an ideal set of features that satisfies all those goals.
3. Identifying goals not met by the features of UniSim.
4. Adding to UniSim the difference between the ideal set of features and the features of UniSim.

The reason for considering other methodologies was that a complex problem appears with features associated with more than one goal.

![Diagram](image)

*Illustration 3: Example of features associated to more than one goal.*

If a feature from the ideal set of features (eg. \( a \)) is associated with several goals (eg. \( g_1, g_2 \)), it may meet a given goal (eg. \( g_1 \)) also met by a feature of UniSim (eg. \( c \)) associated to a different set of goals (eg. \( g_1, g_3 \)). In this situation, the same goal is addressed by two different features. This is undesirable for the final product, since it can lead to a confusing user experience.

A fix could be to iteratively remove features from UniSim and replace them by features from the ideal set, until no more overlapping features left.

As it can be seen, this methodology prioritizes resemblance to the ideal solution over code reuse.
Discovery of differences between SFC and GRAFCET

As it has been said, initially it was assumed that SFC was equivalent to GRAFCET; an SFC editor such as UniSim could serve as basis for our system.

However, during the identification of which stakeholder goals were already met by which features of UniSim (step 1.2 of the methodology by that time), it was revealed that differences between what GRAFCET specifies and what UniSim allowed, were significant.

This discovery demonstrated that the sources of documentation used by that time were not precise enough.

Copies of IEC 60848 and IEC 61131-3 norms were then requested, and in the Annex C of IEC 60848 (Relations between the GRAFCET of IEC 60848 and the SFC of IEC 61131-3) it could be read that:

1. There is no identity between the two graphic representations and the semantics of the two languages.

This allowed to understand that, although they were apparently similar, UniSim and the system needed had substantial differences in their domain models.

Still, UniSim could be partly reused for this project. How it was done, is explained in the chapter about design.

The ultimate methodology

To improve the chances for success, the system was developed by delivering software components early and often [STANDISH].

As a form of Agile Software Development [AGILE], the Scrum methodology [SCRUM] was applied where reasonable.

Since the artifacts defined by Scrum are far from containing all the information required for this report, several artifacts from RUP [RUP] were also generated. Where conflicting in precedence, Scrum was put in practice before, and the resulting experience was reflected in the appropriate RUP artifacts.

The software requirements specification was based on Volere's Requirements Specification Template [VOLERE].

The project had an inception phase, followed by several sprints. At the end of each step, an increment of features was delivered.

<table>
<thead>
<tr>
<th>Inception</th>
<th>Sprint 1</th>
<th>Sprint 2</th>
<th>Sprint ...</th>
<th>Sprint N</th>
</tr>
</thead>
</table>

*Drawing 2: Phases of the project.*

For the inception phase, the next steps were executed:

1. Identification of stakeholders and their goals
2. Business modeling
3. Goals validation and verification
4. Product backlog creation
For each sprint, the next steps were executed:

1. Sprint planning meeting
   1. Product backlog update, if needed
   2. Sprint backlog creation
2. Use case tasks
   1. Use case diagram increment
   2. Use case specification increment
   3. Use case realization increment, if helpful
3. Analysis tasks
   1. Analysis classes diagram increment, if needed
4. Design tasks
   1. Design classes diagram increment, if helpful
   2. Design communication diagram increment, if helpful
   3. Design sequence diagram increment
   4. View of participating design classes
5. Implementation tasks
6. Sprint review meeting
   1. Demo to stakeholders
   2. Feedback gathering from stakeholders

**Evolution of the design tasks**

At the beginning of the project, extensive updates of the design classes diagram and design communication diagrams were performed.

Elaborating these diagrams was valuable since it was unknown what design classes would appear, and how they would interact during each use case.

However, as sprints went on, the design classes diagram became noisy and the design communication diagrams became meaningless.

In the case of the design classes diagram, the source of noise was the multiplication of classes and relationships due to the application of design patterns such as 3-tier architecture, list classes, and others, to the analysis classes.

In the case of the communication diagrams, the loss of meaning was due to the high degree of similarity between diagrams representing different use cases.

The solution for the case of the design classes diagram, was to stop doing this diagram. Doing so without losing information was possible, since the views of participating classes include all the information represented in the classes diagram.

The solution for the case of the communication diagrams, was to stop doing this diagram as well. Doing so without losing information was possible, since the sequence diagrams include all the information represented in the communication diagrams.
Modification from sprint 10

At the end of the sprint 9, the product owner requested a modification over the stated methodology. It was required by him to work first on the use case tasks of all the remaining use cases. So the sprint mechanism would only apply to the analysis tasks onward.
**Business modeling**

The system to be built has to support the GRAFCET standard.

**Use case diagram**

![Diagram](Image)

*Example diagram of a business use case.*

Illustration 4: Business use case diagram.

In the diagram, the user is related to all use cases. Lines from the user to the use cases have been omitted for clarity.
Class diagram

The GRAFCET standard defines several classes, attributes, and relationships between them. Towards a good understanding of these items, having an UML class diagram of them was considered of the greatest interest. A diagram such as this was not found already published, so it was elaborated for this purpose.

GRAFCET

Simplified class diagram

Illustration 5: Simplified business class diagram.
Complete class diagram

Illustration 6: Complete business class diagram.
Requirements specification

The need for good requirements
For professionals from the industrial environment, it may be needed to explain why a good requirement elicitation phase is needed to produce quality software.

Fortunately this can be achieved by only quoting the following paragraph of the book “IEC1131-3 Programming Methodology”.

“The economic side also calls for a stronger role on the design phase [vs development phase]. According to the well known cost progression in quality, the errors due to a bad design, and even worst to an inaccurate requirements specification, are extremely more expensive to be removed than code bugs. Just to fix this concept, we can say that if the cost of removing a code bug is 1, the cost of removing a design mistake is in the order of 10, and it reaches 100 to isolate and solve problems caused by a bad functional (or not functional) problem definition.”

Requirements composition
Requirements can be decomposed in the following manner:

- Functional requirements
- Conceptual schema
- Non-functional requirements

Stakeholders

Current users of GeCé
Engineers that already use the GeCé methodology to program sequential processes in C language. They are familiarized with both GRAFCET and GeCé terminology.

Other GRAFCET designers
Engineers interested in programming sequential processes through GRAFCET but without knowledge enough to program in C language. They are familiarized with GRAFCET terminology, but not with GeCé’s.

Product owner
José Pedro Muñoz is the product owner. He has been nominated by CITCEA to be the gateway with the development team. He has a high level of interest in the project, both as product owner and as a GRAFCET/GeCé user. He has a moderate level of influence in the project. His level of influence is not greater than that because the project is also wanted by other, and higher, management roles in the organization.
The scope of the work

The current situation

The GeCé methodology was developed in year 2005, and since then it has been used regularly by some engineers to manually produce code in C language.

These users perform two activities:

- Diagram drawing: either manually or aided by some already existent software.
- Application of GeCé: manually.

The new situation

The system will allow current users of GeCé and other GRAFCET designers to draw GRAFCET diagrams, and automatically apply the GeCé methodology to generate code in C language.

Thus, the activities of the users will change in the following manner:

- Diagram drawing: Aided by the new system.
- Application of GeCé: Automatically, by the new system.

The context of the work

The system will be used by experts and learners of GRAFCET, running regular computers with Windows. However, in the future it could be required that the system work also on other platforms such as Linux.

Users will be able to exchange files when needed.

Off-the-Shelf solutions

The following table shows the available Off-the-Shelf solutions at the moment of writing, and the stakeholder goals they satisfy.
<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 60848:2002</th>
<th>Verify</th>
<th>C code</th>
<th>Software libre</th>
<th>Cross-platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoDeSys</td>
<td>[Not really] SFC</td>
<td>[Yes]</td>
<td>[No]</td>
<td>[No]</td>
<td>[No]</td>
</tr>
<tr>
<td>Delmia SFC Editor</td>
<td>[Not really] SFC</td>
<td>[Yes]</td>
<td>[No]</td>
<td>[No]</td>
<td>[No]</td>
</tr>
<tr>
<td>FluidSIM</td>
<td>[Yes]</td>
<td>[Yes]</td>
<td>[No]</td>
<td>[No]</td>
<td>[No]</td>
</tr>
<tr>
<td>Grafed</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
<td>[Yes] Java</td>
</tr>
<tr>
<td>GRAFTOR</td>
<td>[Not really] Year 2000</td>
<td>[No]</td>
<td>[No]</td>
<td>[Yes]</td>
<td>[No] Just Linux</td>
</tr>
<tr>
<td>ISaGRAF</td>
<td>[Not really] SFC</td>
<td>[Yes]</td>
<td>[Yes]</td>
<td>[No]</td>
<td>[No]</td>
</tr>
<tr>
<td>JGrafchart</td>
<td>[Not really] SFC</td>
<td>[Yes]</td>
<td>[No]</td>
<td>[No] Freeware</td>
<td>[Yes] Java</td>
</tr>
<tr>
<td>SFCEDIT</td>
<td>[Yes]</td>
<td>[Yes]</td>
<td>[No]</td>
<td>[No]</td>
<td>[No]</td>
</tr>
<tr>
<td>UniSim</td>
<td>[Not really] SFC</td>
<td>[Yes]</td>
<td>[No]</td>
<td>[Yes] GPL</td>
<td>[Not really] .NET</td>
</tr>
</tbody>
</table>

*Table 1: Goals satisfied by available off-the-shelf solutions.*
The following table shows the references where this information can be found.

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delmia SFC Editor</td>
<td><a href="http://www.3ds.com/products/delmia/portfolio/delmia-v6/all-products/domain/Factory_Definition_Simulation/product/LE1/">http://www.3ds.com/products/delmia/portfolio/delmia-v6/all-products/domain/Factory_Definition_Simulation/product/LE1/</a></td>
</tr>
<tr>
<td>FluidSIM</td>
<td><a href="http://www.fluidsim.de/fluidsim/indexhb4_e.htm">http://www.fluidsim.de/fluidsim/indexhb4_e.htm</a></td>
</tr>
<tr>
<td>Grafed</td>
<td><a href="http://dev.sorcix.com/">http://dev.sorcix.com/</a></td>
</tr>
<tr>
<td>ISaGRAF</td>
<td><a href="http://www.isagraf.com/pages/products/Isagraf/appworkbench.htm">http://www.isagraf.com/pages/products/Isagraf/appworkbench.htm</a></td>
</tr>
<tr>
<td>JGrafchart</td>
<td><a href="http://www.control.lth.se/Research/Tools/grafchart/">http://www.control.lth.se/Research/Tools/grafchart/</a></td>
</tr>
</tbody>
</table>

Table 2: References for off-the-shelf solutions.

An analysis of the above solutions, shows that none of them satisfies all the goals of the stakeholders. However, there are two with a license which allows customization: GRAFTOR, and UniSim.

A comparative of these two on few relevant aspects, allows to conclude that UniSim provides a more interesting base to build the new solution on.

<table>
<thead>
<tr>
<th></th>
<th>UniSim</th>
<th>GRAFTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year of last update</td>
<td>2007</td>
<td>2000</td>
</tr>
<tr>
<td>IDE</td>
<td>Visual Studio</td>
<td>Visual Tcl</td>
</tr>
<tr>
<td>Object-oriented</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Body hierarchy</td>
<td>Macro-steps</td>
<td>None</td>
</tr>
<tr>
<td>Actions</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Variables</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3: UniSim vs GRAFTOR.
Illustration 7: Screenshot of UniSim.

Illustration 8: Screenshot of GRAFTOR (although the title reads “Grafedit”).
Functional requirements

Product use case list

In the diagram, the user is related to all use cases. Lines from the user to the use cases have been omitted for clarity.

Individual product use cases
New project

Primary actor: User.
Precondition: None.
Trigger: The user wants to start a new project.
Main success scenario:
1. The user selects to create a new project.
2. The user selects a destination folder.
3. The user gives a name for the project.
Extensions:
2a. A project with the same name exists in the same folder already.
   2a1. The user gives another name for the project.
2a. A project with the same name exists in the same folder already.
   2a1. The user selects another destination folder for the project.
Customer satisfaction: 2
Customer dissatisfaction: 5
Dependences: None.
Supporting materials: None.
Originator: User.

Save project

Primary actor: User.
Precondition: None.
Trigger: The user wants to save the current project.
Main success scenario:
1. The user selects to save.
Extensions: None.
Customer satisfaction: 2
Customer dissatisfaction: 4
Dependences: New project.
Supporting materials: None.
Originator: User.
Open project
Primary actor: User.
Precondition: None.
Trigger: The user wants to open a saved project.
Main success scenario:
1. The user selects to open a project.
2. The user gives the project's path.
Extensions:
2a. The path is not the one of a valid project.
2a1. The user gives another path for the project.
Customer satisfaction: 2

New grafcet
Primary actor: User
Precondition: None.
Trigger: The user wants to add a grafcet to the current project.
Main success scenario:
1. The user selects to add a new grafcet to the current project.
2. The user gives a name for the grafcet.
Extensions:
2a. A grafcet with the same name exists already.
2a1. The user gives another name for the grafcet.
Customer satisfaction: 1
Customer dissatisfaction: 5
Dependencies: New project.
Supporting materials: None.
Originator: User.
Customer dissatisfaction: 4
Add initial step

Primary actor: User.
Precondition: None.
Trigger: The user wants to add an initial step.
Main success scenario:
1. The user selects to add an initial step.
2. The user gives a name for the initial step.
3. The user places the initial step on the diagram.
Extensions:
2a. A step with the same name exists already.
2a1. The user gives another name for the initial step.
Customer satisfaction: 1
Customer dissatisfaction: 5
Dependencies: New grafcet.
Supporting materials: None.
Originator: User.

Add step

Primary actor: User.
Precondition: None.
Trigger: The user wants to add a step.
Main success scenario:
1. The user selects to add a Step.
2. The user gives a name for the Step.
3. The user places the step on the diagram.
Extensions:
2a. A step with the same name exists already.
2a1. The user gives another name for the step.
Customer satisfaction: 1
Customer dissatisfaction: 5
Dependencies: New grafcet.
Supporting materials: None.
Originator: User.
Add project variable

Primary actor: User.

Precondition: None.

Trigger: The user wants to add a project variable.

Main success scenario:
1. The user selects to add a project variable.
2. The user selects the scope of the variable.
3. The user gives a name for the variable.
4. The user selects the type of the variable between BOOL, INT, and REAL, as defined by IEC61131-3.
5. The user selects the application of the variable between input, internal, and output.
6. The user checks none or some of the variable qualifiers: register, static, volatile.
7. The user sends the variable's data to the system.

Extensions:
2a. A variable with the same name already exists.
   2a1. The user gives another name for the variable.

Customer satisfaction: 2
Customer dissatisfaction: 5

Dependencies: None.

Supporting materials: None.

Originator: User.
Add grafcet variable

Primary actor: User.

Precondition: None.

Trigger: The user wants to add a grafcet variable.

Main success scenario:
1. The user selects to add a project variable.
2. The user selects the scope of the variable.
3. The user gives a name for the variable.
4. The user selects the type of the variable between BOOL, INT, and REAL, as defined by IEC61131-3.
5. The user selects the application of the variable between input, internal, and output.
6. The user checks none or some of the variable qualifiers: register, static, volatile.
7. The user sends the variable's data to the system.

Extensions:
2. A variable with the same name already exists.
   2A1. The user gives another name for the variable.

Customer satisfaction: 2

Customer dissatisfaction: 5

Dependencies: None.

Supporting materials: None.

Originator: User.
Add transition

**Primary actor**: User.

**Precondition**: None.

**Trigger**: The user wants to add a transition.

**Main success scenario**: 
1. The user selects to add a Transition.
2. The user gives a condition for the transition.
3. The user places the transition on the diagram.

**Extensions**: 
2a. The given condition is not well-formed.
   2a1. The user gives another condition for the transition.
2b. The user wants to choose a variable from the variables list.
   2b1. The user selects to view the variables list.
   2b2. The user chooses a variable from the variables list.
   2b2a. The user wants to declare a new variable.
      2b2a1. The user declares a new variable.
2c. The given condition contains one or more not declared variables.
   2c1. The system asks the user to declare the first not declared variable
   2c2. The user declares that variable.

**Customer satisfaction**: 1

**Customer dissatisfaction**: 5

**Dependencies**: Add step.

**Supporting materials**: None.

**Originator**: User.

**Dependencies**: New project.
**Code generation**

**Primary actor:** User.

**Precondition:** None.

**Trigger:** The user wants a code in C language that behaves such as the drawn diagram.

**Main success scenario:**
1. The user selects to generate C code.
2. The system outputs code that behaves as the specified by the user in the diagram, and follows the GeCé methodology.

**Extensions:** None.

**Customer satisfaction:** 5

**Customer dissatisfaction:** 4

**Dependencies:** Add step, add transition.

**Supporting materials:** GeCé methodology.

**Originator:** User.

**Run**

**Primary actor:** User.

**Precondition:** None.

**Trigger:** The user wants to verify a grafcet design.

**Main success scenario:**
1. The user selects to run.

**Extensions:** None.

**Customer satisfaction:** 5

**Customer dissatisfaction:** 2

**Dependencies:** Add step, add transition, code generation.

**Supporting materials:** None.

**Originator:** User.
Add action

**Primary actor:** User.  
**Precondition:** None.  
**Trigger:** The user wants to add an action to a step.  

**Main success scenario:**  
1. The user selects a step.  
2. The user selects to add an action.  
3. The user enters action's data.  

**Extensions:** None.  
**Customer satisfaction:** 4  
**Customer dissatisfaction:** 3  
**Dependencies:** Add step.  
**Supporting materials:** None.  
**Originator:** User.  

Add Macro-step

**Primary actor:** User.  
**Precondition:** None.  
**Trigger:** The user wants to add a Macro-step.  

**Main success scenario:**  
1. The user selects to add a Macro-step.  
2. The user gives a name for the Macro-step.  
3. The user places the Macro-step on the diagram.  
4. The user defines an expansion for the Macro-step.  

**Extensions:**  
2a. A step with the same name exists already.  
2a1. The user gives another name for the Macro-step.  

**Customer satisfaction:** 4  
**Customer dissatisfaction:** 2  
**Dependencies:** Add step.  
**Supporting materials:** None.  
**Originator:** User.
Add Enclosing Step

Primary actor: User.

Precondition: None.

Trigger: The user wants to add an Enclosing Step.

Main success scenario:
1. The user selects to add an Enclosing Step.
2. The user gives a name for the Enclosing Step.
3. The user places the Enclosing Step on the diagram.
4. The user selects to add an enclosure to the Enclosing Step.
5. The user enters enclosure's data.

Extensions:
2a. A step with the same name exists already.
   2a1. The user gives another name for the step.
3a. A grafcet with the same name exists already.
   3a1. The user gives another name for the enclosure.

Customer satisfaction: 3
Customer dissatisfaction: 1
Dependencies: Add step.
Supporting materials: None.
Originator: User.
Add Forcing Order

Primary actor: User.

Precondition: None.

Trigger: The user wants to add a Forcing Order.

Main success scenario:
1. The user selects a step.
2. The user selects to add a Forcing Order.
3. The user enters forcing action's data.

Extensions: None.

Customer satisfaction: 3

Customer dissatisfaction: 2

Dependencies: Add step.

Supporting materials: None.

Originator: User.

Supporting materials: None.

Originator: User.
In the diagram, only the most significant attributes of the classes are included. The rest are omitted for clarity.
**Changes over the business class diagram**

- **Added classes**
  - MacroStepBody
    - Represents the expansion of macro-steps.
    - It is created to avoid “MacroStep” to inherit from “BaseStep”, and “Body”, which although conceptually possible, would be harder to understand.
  - Body
    - Contains the common traits between “Grafcet” and “MacroStepBody”
  - Enclosure
    - Created to be used instead of “Grafcet” for EnclosingStep, which although conceptually and technically possible, would be harder to understand.
  - BaseStep
    - Contains the common traits between “Step” and “MacroStep”.
  - EnclosedStep
    - Avoids having to place the attribute “activationLink” on Step.
    - This gives a bit more semantic fidelity in exchange for an additional class.
  - BooleanVariable, IntegerVariable, RealVariable
    - The business class “Variable” contains structural differences depending on the data type. These new classes are used to represent more precisely those differences.

- **Modified classes**
  - GlobalGrafcet: Now is “Project”.
    - The change of name is due to the terminology employed by the Product Owner.
  - PartialGrafcet: Now is “Grafcet”.
    - The change of name is due to the terminology employed by the Product Owner.
  - Step
    - The “initial” attribute is created to replace the “InitialStep” class of the business model.
    - This change avoids having to create also “Initial” versions of EnclosingStep and EnclosedStep, which although conceptually possible, would result in additional clutter for the diagram in exchange for very little semantic gain.
  - StoredAction
    - Losses its association with transition.
    - This change is due to using this association being considered as a bad practice by one expert GRAFCET designer.
  - Variable
The “application” attribute is created to replace the “InputVariable”, “InternalVariable”, and “OutputVariable” classes of the business model.

This simplification removes clutter from the class diagram without hurting perceptibly its semantics, since all those classes had the same structure and behavior.

- Removed classes
  - DelayedCondition
    - It is now represented by a particular case of TimeDependentCondition.
  - EventCondition
    - It is not used by GRAFCET designers.
Non-functional requirements

Usability and humanity requirements

Learning requirements
Each feature relevant to a given user of the profiles “Current GeCé user”, and “Other GRAFCET designers”, should be learned by her within less than 10 minutes, by simply using the system.

Maintainability and support requirements

Adaptability requirements
The system is expected to run under Windows XP and Windows 7.
The system might eventually be asked to run under Linux. At that moment, the system should be adapted within less than 24 hours.

Legal requirements

Standards Requirements
The system shall comply with IEC60848 ed2.0 standard.
The system should comply with successive editions of IEC60848 standard.
The system should export diagrams to an XML-based format.

Satisfaction argument
Following there is a relationship between goals and the system features that satisfy those goals.

<table>
<thead>
<tr>
<th>Goal</th>
<th>System features</th>
</tr>
</thead>
<tbody>
<tr>
<td>To create, edit, and visualize grafcets without the need for additional training.</td>
<td>The graphic editor, with each feature learnable in less than 10 minutes.</td>
</tr>
<tr>
<td>To verify the design of the diagram.</td>
<td>The ability to run the designed grafcets.</td>
</tr>
<tr>
<td>To exchange grafcets with other users in an interoperable format.</td>
<td>Export and import diagrams in an XML-based format.</td>
</tr>
<tr>
<td>To translate these grafcets to C following the GeCé methodology.</td>
<td>The ability to generate a code that follows the GeCé methodology.</td>
</tr>
<tr>
<td>All together by focusing on the intrinsic value of the grafcets, and by benefiting from a graphical representation with which they are familiarized; without the need for analyzing, writing, reading or interpreting C code.</td>
<td>The graphic editor, and the adherence to the IEC60848 standard.</td>
</tr>
</tbody>
</table>

Table 4: Satisfaction argument.
**Planning**

The first month of the project was devoted to gaining knowledge about the domain. It was necessary to achieve different levels of comprehension about different issues. Following is listed which issues were those, which level of understanding was needed, and why.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Level of understanding</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAFCET basic rules</td>
<td>High</td>
<td>GRAFCET is part of the fundamentals of the product.</td>
</tr>
<tr>
<td>GeCé</td>
<td>Medium</td>
<td>The diagram editor had to produce something that, through more or less transformation, was a valid input for code generation.</td>
</tr>
<tr>
<td>Gemma Guide</td>
<td>Medium</td>
<td>A need initially mentioned by stakeholders, was to precede the definition of GRAFCET diagrams with Gemma Guides. An introduction to Gemma Guides is available at <a href="http://edison.upc.edu/curs/grafcet/gemma/presenta.html">http://edison.upc.edu/curs/grafcet/gemma/presenta.html</a>.</td>
</tr>
<tr>
<td>UniSim architecture</td>
<td>High</td>
<td>If the product had to reuse UniSim, this could only be done from a precise view of what was reusable and what was not.</td>
</tr>
</tbody>
</table>

*Table 5: Domain knowledge needed for the project.*

Interestingly enough, what at the beginning was considered just as “training”, became a considerable part of the business modeling activities.

The training tasks were followed by the inception phase of the project.

The inception phase was followed by sprints.

A preliminary report had to be delivered by the end of April.

The development tasks were scheduled to end by the end of May, and devote the remaining time to the writing of this report.
Illustration 12: Planification at the beginning of the project.
As weeks passed, it was realized that the artifacts elaborated during the sprints, were in a state advanced enough to shorten the period needed for writing this report. This gave three extra weeks. However, knowledge transfer to the IT staff of CITCEA had not been taken into account. This took one week.

Considering all, the chronology of the tasks was the following.
As it can be seen, sprint 17 was interrupted by the need for knowledge transfer.
**Minor sprints**

Something foreseeable although not foreseen occurred during sprints, was that stakeholders could be unsatisfied by some aspects of the demoed features. This was addressed by performing minor sprints devoted to rectify the work done on the previous regular sprint. To make a clearer distinction between regular and minor sprints, a different notation was used: “Sprint X” for regular sprints, and “Sprint X.Y” for minor sprints.

*Illustration 14: Sprint feedback as seen at the beginning (above) and at the end (below) of the project.*
Features by sprint

Which features were implemented during which sprints can be seen in the following table.

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature</th>
<th>Sprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edition</td>
<td>Regular step</td>
<td>1</td>
</tr>
<tr>
<td>Edition</td>
<td>Initial step</td>
<td>1</td>
</tr>
<tr>
<td>Edition</td>
<td>Transition without condition</td>
<td>1</td>
</tr>
<tr>
<td>Compilation</td>
<td>Step compilation</td>
<td>1</td>
</tr>
<tr>
<td>Compilation</td>
<td>Transition compilation</td>
<td>1</td>
</tr>
<tr>
<td>Structuring</td>
<td>Multiple grafcets</td>
<td>2</td>
</tr>
<tr>
<td>Transitions</td>
<td>Transition with condition: Boolean predicate</td>
<td>3</td>
</tr>
<tr>
<td>Interaction</td>
<td>Several enhancements</td>
<td>3.1</td>
</tr>
<tr>
<td>Transitions</td>
<td>Transition with condition: time after rising edge of step variable</td>
<td>4</td>
</tr>
<tr>
<td>Verification</td>
<td>Compilation of C code</td>
<td>5</td>
</tr>
<tr>
<td>Verification</td>
<td>Execution of the generated code</td>
<td>5</td>
</tr>
<tr>
<td>Verification</td>
<td>Communication with the executed code: read something</td>
<td>5</td>
</tr>
<tr>
<td>Verification</td>
<td>Communication with the executed code: write something</td>
<td>5</td>
</tr>
<tr>
<td>Verification</td>
<td>Read active steps</td>
<td>5</td>
</tr>
<tr>
<td>Verification</td>
<td>Order grafcet evolution</td>
<td>5</td>
</tr>
<tr>
<td>Sequencing</td>
<td>Activation of parallel sequences</td>
<td>6</td>
</tr>
<tr>
<td>Sequencing</td>
<td>Synchronization and activation of parallel sequences</td>
<td>6</td>
</tr>
<tr>
<td>Verification</td>
<td>Set variable values</td>
<td>7</td>
</tr>
<tr>
<td>Interaction</td>
<td>Several enhancements</td>
<td>7.1</td>
</tr>
<tr>
<td>Actions</td>
<td>Continuous action without condition</td>
<td>10</td>
</tr>
<tr>
<td>Interaction</td>
<td>Several enhancements</td>
<td>10.1</td>
</tr>
<tr>
<td>Interaction</td>
<td>Several enhancements</td>
<td>10.2</td>
</tr>
<tr>
<td>Actions</td>
<td>Stored action: on rising/falling edge of step variable</td>
<td>11</td>
</tr>
<tr>
<td>Structuring</td>
<td>Macro-step</td>
<td>12</td>
</tr>
<tr>
<td>Structuring</td>
<td>Forcing of a partial grafcet</td>
<td>13</td>
</tr>
<tr>
<td>Structuring</td>
<td>Enclosure</td>
<td>14</td>
</tr>
<tr>
<td>Persistence</td>
<td>Save grafcets to disk</td>
<td>15</td>
</tr>
<tr>
<td>Persistence</td>
<td>Load grafcets from disk</td>
<td>15</td>
</tr>
<tr>
<td>Actions</td>
<td>Continuous action with condition: time dependent assignation condition</td>
<td>16</td>
</tr>
<tr>
<td>Actions</td>
<td>Continuous action with condition: Boolean predicate</td>
<td>17</td>
</tr>
<tr>
<td>Transitions</td>
<td>Transition with condition: time dependent transition-condition</td>
<td>17</td>
</tr>
</tbody>
</table>

Illustration 15: Features by sprint.
**Analysis**

**Code generation**

As stated by the GeCé methodology in its chapter “Definició de variables i constants”, the code generation use case needs from a mapping between elements in the diagram and the GeCé variables that represent those elements in the C code.

So, the code generation use case is solved in two stages:

1. Map generation
2. Actual code generation

None of these stages require additional analysis beyond what is already specified in the GeCé methodology, except for the case of macro-steps.

**Macro-step code generation**

The way specified by GeCé to generate the code corresponding to macro-steps is to integrate them expanded into the graftet.

Expanding a macro-step consists in promoting the steps of its expansion as a set of steps that replaces the macro-step. This has complex effects on the associations between steps and transitions.

For the particular case of the current product, this expansion impacts both two stages of the verification sequence: Map generation, and code generation. In these stages, macro-steps will be treated as if their internal sequences were expanded in the main graftet; replacing its container macro-step.

**Map generation**

Without macro-steps, map generation would be trivial.

---

Illustration 16: Excerpt from Committee Draft IEC 60848 Ed. 2: Expansion of the macro-step

The way specified by GeCé to generate the code corresponding to macro-steps is to integrate them expanded into the graftet.
• For steps: It would be achieved by simply going iteratively through the steps of the grafcet and assigning unique numbers to them.

• For transitions: Just as for steps.

Illustration 17: Mapping between steps of the diagram and steps of the C code.

But with macro-steps, the previous method fails due to the following reasons:

• It assigns a number to macro-steps, which should be ignored; not assigning a number.
• It does not assign a number to the steps of the expansion of macro-steps.
• It does not assign a number to the transitions of the expansion of macro-steps.

Illustration 18: Wrong mapping between steps of the diagram and steps of the C code.

A way to overcome these limitations is to enhance those iterative algorithms in such a way that, when they find a macro-step, behave with the following changes:
• Don't assign a number to the macro-step.
• Enter into the macro-step, and go through their internal steps.

Illustration 19: Correct mapping between steps of the diagram and steps of the C code.

Code generation

Expansion of macro-steps affects the generation of the GeCé's function EvolucioGrafzet. During the generation of the code for this function, associations between steps and transitions are accessed in three different ways:

• Getting transitions by previous step
• Getting steps by next transition
• Getting steps by previous transition

Without macro-steps, accessing to this information would be trivial. It would be only needed to access, respectively, attributes “downTransitions”, “upSteps”, and “downSteps” already existent in the GRAFCET conceptual schema class diagram.

Illustration 20: Detail of steps and transitions, from the conceptual schema class diagram.

But with macro-steps, the previous methods fail due to the following reasons:
• “downTransitions” does not contain the transitions that should follow the expansion of a macro-step: the transitions after the macro-step.
• “upSteps” does not contain the steps that end the expansion of a macro-step.
• “downSteps” does not contain the steps that start the expansion of a macro-step.

So for example, in the previous illustration:
• Getting transitions by previous step would fail for S3, not returning h.
• Getting steps by next transition would fail for h, not returning S3.
• Getting steps by previous transition would fail for a, not returning E3.

A way to overcome these limitations, is to define higher level operations that take into account whether some of the involved steps is a macro-step or not, and use these operations for code generation, instead of the bare associations.

• Get transitions by previous step: If the step is the end step of a macro-step's expansion (eg. S3), down-transitions of the macro-step (eg. h) become down-transitions of the step.
• Get steps by next transition: If a step that precedes the transition (eg. h) is a macro-step, the exit step of the macro-step's expansion (eg. S3) becomes up-step of the transition.
• Getting steps by previous transition: If a step that follows the transition (eg. a) is a macro-step, initial steps of the macro-step's expansion become down-steps of the transition (eg. E3).
Verification

Verification of grafcets can be done through two ways:

1. A simulation engine that evolves the state of a virtual system by applying evolution rules to the defined grafcets.
   - Pros:
     - It does not require the use of additional technologies.
   - Cons:
     - There can be differences between the behavior of the simulation engine and the behavior of the actual generated code, which can lead to verification errors.

2. Compilation, execution, and control of the already generated code.
   - Pros:
     - It reuses the generated code.
     - It could be reusable to monitor the execution of the grafcet on the actual, industrial system, which is a valuable feature for stakeholders.
   - Cons:
     - It requires a C compiler accessible to the product.
     - It requires a communications infrastructure on both sides: the product and the controlled code.

In this project, the second way will be explored. Thus, the following actions will be required:

1. Generation of a map between domains, in both directions: one for generating the code and make requests to the controlled code, and one for processing the responses of the generated code.
2. Generation of executable code.
3. Compilation of executable code.
4. Execution.
5. Process monitoring and control.

A graphical view detailing the inputs and outputs of these actions is the following:
An initial assignment of responsibilities to classes for the above sequence is the following.

Illustration 22: Input and output of grafcet verification actions.
Illustration 23: Initial assignment of responsibilities for grafcet verification.
Design

Design goals

Regarding design, there are two initial goals for the product:

- Using a three-tier architecture, with the following tiers: Presentation, domain, and infrastructure.
- Low coupling and high cohesion.

However, UniSim does not follow the three-tier architecture, but one that strictly portrays the domain model of IEC 1131 (see below), assigning presentation, domain, and persistence responsibilities to the same classes.

To deal with this conflict, the following rules have been applied:

1. Reused classes with presentation responsibilities will be located in presentation tier.
2. The remaining classes will be located into its corresponding tier: presentation, domain, or infrastructure.

Reusing UniSim

As we have explained, we found that UniSim could be partially reused to build the product.

So it was necessary to:

1. Determine which parts of it seemed convenient to reuse.
2. Model the architecture of those parts, to understand how the product could reuse them.
3. Verify the model obtained in step 2, by extracting those parts from UniSim, and get them working independently.

Finding the reusable components

Three main components were identified in UniSim:

1. The graphical user interface (GUI)
2. The domain model
3. The simulator

Illustration 24: UniSim components.
The GUI

The GUI of the SFC editor was interesting by two reasons:

1. SFC diagrams are visually similar to GRAFCET diagrams
   1. Steps are identical
   2. Transitions are identical
   3. Actions differ slightly
2. Its interaction model was quite user-friendly
The domain model

As it can be seen in the illustration below, there are radical and extensive differences between the IEC 61131 software model and IEC 60848's. Nevertheless, some low level classes related to variables are pretty similar.

Illustration 25: Domain model of IEC 1131, containing SFC (source: DISANZO).
The simulator
The simulator of UniSim is not of interest since the product will not really simulate the dynamics of the grafcet. It will, instead, allow to verify the grafcet through monitoring and control of a program obtained through the application of the GeCé methodology.

**Design model of the reusable components**

The GUI
Classes involved in the GUI of the SFC editor and their dependencies are shown in the following diagram.

Legend:
- Blue rectangles reflect the folder structure.
- Discontinuous arrows note loose coupling.
Considering the diagram above, the product could simply create an instance of FormSfc to reuse the full GUI of the SFC editor.

Dependencies of the SFC editor's GUI with the SFC domain model should be replaced by associations with GRAFCET domain model classes.

**GUI design model verification**

The previously identified SFC editor GUI classes were created in a new Windows Forms Application for .NET 2.0 written in Visual Basic 2008.

The code of the classes was almost copy pasted. Only automatically generated code from forms was excluded.

Compilation was then attempted. Near one hundred errors were found. The cause of all these errors was that coupling with “Pous” and “Pou” (Program Organization Unit, in SFC) classes was stronger than initially realized.

However this made sense since the GUI access the domain model through these classes.

After the removal of references to “Pous” and “Pou” classes, and the addition of several auxiliary (and initially omitted) classes from “dataTypes” and “pous/commonObjects” folders, the compilation succeeded.

The application was then successfully launched. A simple diagram could then be created.

*Illustration 27: SFC editor GUI working out of UniSim*
Cross-platform design

UniSim is programmed in Visual Basic, using .NET Framework 1.1.

<table>
<thead>
<tr>
<th>.NET Version</th>
<th>Public data</th>
<th>Private data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0 [XMLSERIALIZER]</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>3.0 [DATACONTRACT]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>3.5 [DATACONTRACT]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>4.0 [DATACONTRACT]</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Table 6: Automatic serialization and deserialization capabilities by .NET Framework version.*

On the other hand, the only working, updated, cross-platform implementation of the .NET Framework is Mono.

However the newest version of Visual Basic supported by Mono is that corresponding to .NET Framework 2.0.
Considering also the cross-platform requirements specified for this product (see “The context of the work” section, on “Requirements” chapter), this project upgrades the original code from UniSim to .NET Framework 2.0.

<table>
<thead>
<tr>
<th>.NET Version</th>
<th>Visual Basic</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2.0</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>3.0</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>3.5</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>4.0</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 7: Mono language support by .NET Framework version.

A future without Visual Basic

As with many other software systems, keeping the product useful, secure, and cost-effective will probably demand keeping its technology stack within an age limits. So the .NET Framework version used by the product will probably have to be upgraded from time to time.

If arrived the time of upgrading the product from .NET Framework 2.0, Mono still does not support a newer version of Visual Basic, a change of language may be the best solution.

For now, Mono does support recent versions of C#, so the simplest change could be creating a copy of the product on C#, and upgrading to a recent version of .NET from there. Several code conversion tools available online [CARLOSAG] [DEVELOPERFUSION] [TELERIK] can speed up this process.
Code generation

Actions
During actions, variables that represent the state of the controlled system are read or written. However, to avoid computational inconsistencies, those variables tied to inputs or outputs of the controlled system, shall not be accessed directly, but through an image table which is updated at the beginning of each scan cycle, and dumped at the end of it. This will improve the chances of doing the calculations based on a single snapshot of the state of the controlled system. [BOIX]

The GeCé methodology specifies the use of the following data structures to create this image tables.

- For digital inputs: A vector named "EntradaDig", where each position corresponds to a Boolean input.
- For analog inputs: The creation of a variable whose name is prefixed with "EA_", for each analog input.
- For digital outputs: A vector named "SortidaDig", where each position corresponds to a Boolean output.
- For digital outputs: The creation of a variable whose name is prefixed with "SA_", for each analog output.

The need for using images instead of the actual variables, forces users of the GeCé methodology to define actions using the names of the images, instead of the names of the actual values.

A feature of the product is doing this translation job for them; automatically.

Illustration 29: Dynamics of the image table.

Illustration 30: Conceptual view of data flow of action translation.

Through lexical analysis of the actions defined by the user, the product will detect the names of input variables and output variables, and then it will translate these names to their corresponding image names.

More in specifically, the process will consist in two steps:
1. Lexical analysis of the actions written by the user, produces tokens.
2. For each identifier found among those tokens, if it is the name of an input variable or an output variable, replace it by its corresponding image name. This produces action code written as specified by GeCé.

Illustration 31: Conceptual view of data flow of action translation in two steps.

Lexical analysis is performed by lexical analyzers (also called lexers, or scanners). Towards building a lexer for the productl, a generic lexer is observed first.

Illustration 32: Data flow of a generic lexer.

The grammar given to the lexer, and the language the lexer is written in, will depend on the particular problem to solve. For our case, these will be:

- Grammar: The grammar of the C language.

The task of building a lexer is hard, but fortunately several lexer generators are publicly available. Towards picking and using a lexer generator, a generic one is observed first.

Illustration 33: Data flow of a generic lexer generator.

One of these generators able to create the lexer in VB.NET is Coco/R. [COCO] Coco/R accepts grammars described in Extended Backus–Naur Form (EBNF).
ANSI C syntax written in EBNF is available at http://www.cs.man.ac.uk/~pjj/bnf/c_syntax.k_r, which can be accessed from http://www.cs.man.ac.uk/~pjj/bnf/bnf.html (Notations for context-free grammars).

Once the lexer is built, it can be used in the product.

Illustration 34: Technical view of data flow of the lexer generator used for the product.

Illustration 35: Technical data flow of action translation.
Verification

During the verification use case, communications are needed to interchange data between the product and the process running the generated code (here named "executable").

For this project, communications will be based on sockets technology, given that it allows not only communications within the same computer, but also between computers in the same network. This could eventually be used to monitor the execution of the GRAFCET on remote machines.

Through a socket, messages will be sent between the product and the executable.

Data accessed will be only that of the executable. Thus in communications terms, the executable will play the role of a server, and the product the role of a client.

There are two types of information that client and server have to share: steps status, and variables values.

Steps status

The only specific need for communicating the status of the steps is getting the status of all steps. The mechanism to identify which steps are being referred in the messages, is by position.

For instance, to get the status of all steps, given a total of four steps with status inactive, active, inactive, and inactive respectively:

- The client will send this message: ReqGetStepStatus
- The server will respond with this message: ResGetStepStatus 0100.

This protocol implies that:
• When the server responds, it has to send the status of all steps in order.
• When the client processes the response, it has to identify the step from its position.

To implement these tasks, the following techniques are used:
• On the server: The status of the steps is stored into a C unsigned int variable, where each bit represents a different step.
• On the client: A mapping between the position of the bit in the server's variable, and the actual step in the domain model, is stored into a VB Dictionary Of Integer, BaseGraphicalStep.

**Variables value**

Specific needs for communicating the values of the variables are:
• Setting one variable value
• Getting all variables value

The mechanism to identify which variable or variables are being referred in the messages, will depend on whether the message refers to one or all variables.
• One variable: Identification by name.
• All variables: Identification by position.

For instances:
• To set the value 300 for a variable named “engine1”:
  ◦ The client will send this message: ReqSetVarValue engine1 300.
  ◦ The server will confirm the reception with: ResSetVarValue engine1 300.

*Drawing 9: Communications for variables value setting.*
To get the value of all variables, given a total of four variables with values 300, 2.3, 12, and 145 respectively:

- The client will send this message: ReqGetVarValues
- The server will respond with this message: ResGetVarValues 300,2.3,12,145

This protocol implies that, depending on the ongoing transaction and its moment, the client and the server have to perform different tasks.

<table>
<thead>
<tr>
<th>Moment</th>
<th>Setting one variable</th>
<th>Getting all variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client constructs the request</td>
<td>Codify the variable name to a string</td>
<td>-</td>
</tr>
<tr>
<td>Server processes the request</td>
<td>Identify the variable from a string</td>
<td>Send all variables in order</td>
</tr>
<tr>
<td>and responds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client processes the response</td>
<td>Identify the variable from a string</td>
<td>Identify the variable from its position</td>
</tr>
</tbody>
</table>

*Table 8: Tasks by verification transaction.*
To implement this tasks, the following techniques were initially considered.

<table>
<thead>
<tr>
<th>Moment</th>
<th>Setting one variable</th>
<th>Getting all variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client constructs the request</td>
<td>Using a VB Dictionary Of Variable, Integer. The integer represents the position of the variable in the array used by the server.</td>
<td>-</td>
</tr>
<tr>
<td>Server processes the request</td>
<td>Using a C array of pointers to the variables, by position.</td>
<td>The array is went across, writing all values.</td>
</tr>
<tr>
<td>and responds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client processes the response</td>
<td>Using a VB Dictionary Of Integer, Variable, mirror from the above one.</td>
<td>The response is read while accessing the dictionary Of Integer, Variable by ascending key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9: Techniques initially considered for verification tasks.

Regarding the C array of pointers to variables by position, since these variables could be of different data types (integer, float, ...), the data type considered for the array was “void”.

However, this array presented technical difficulties at the associations between values and the variables. In particular, these associations provoked the error "illegal indirection" from the compiler. The cause, apparently, is that it is not possible to read or write a value pointed by a void pointer, even using type casting.

Given the constraint of not being able to use arrays of pointers to void, another tactic was considered.

Instead of making the server to process the request from these elements:

- An array "A" of N positions, being N the number of variables.
- A position in the array "p".
- An instruction that accesses to "A[p]".

The request could be processed using the following:

- N instructions of access; one for each variable.

Pros and cons of this alternative were studied technically and conceptually.

Technically:

- Pro: Since the role of an intermediary played by the array is not needed, the array can be suppressed.
- Apparent con: It seems that the length of the resulting program is greater because of the N assignation instructions instead of 1. But it also has to be accounted that using the array implies initializing it with N assignations; one for each variable.

Conceptually:

- Pros: None.
- Cons: Since the responsibility of which variable has to be accessed is moved from the
generated code to the generator code, one degree of dynamism in that regard is lost. Nonetheless, this latter con has no effect over the quality of the solution, since it does not hinder any of its requirements.

The resulting table of techniques by task is the following.

<table>
<thead>
<tr>
<th>Moment</th>
<th>Transaction set one variable</th>
<th>Transaction get all variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client constructs the request</td>
<td>The <strong>name</strong> of the variable is used to codify the variable in the message.</td>
<td>-</td>
</tr>
<tr>
<td>Server processes the request</td>
<td>A <strong>switch</strong> selects a hard-coded association between the value and the (hard-coded) variable.</td>
<td>In-order, hard-coded set of “send” <strong>instructions</strong>, one for each variable.</td>
</tr>
<tr>
<td>Client processes the response</td>
<td>An <strong>VB Dictionary</strong> of String, Variable. The string is used to get the variable.</td>
<td>An <strong>VB Dictionary</strong> of Integer, Variable. The integer represents the position of the variable in the set of “send” instructions executed by the server. The response is read while accessing the dictionary by ascending key.</td>
</tr>
</tbody>
</table>

*Table 10: Adopted techniques by verification task.*
Adding new data types

The product allows defining variables that can be accessed by conditions and actions. GRAFCET does only make explicit distinction between Boolean and non-Boolean variables. The data types that this project makes available are three: BOOL, INT, REAL, as defined by IEC61131-3.

<table>
<thead>
<tr>
<th>No.</th>
<th>Keyword</th>
<th>Data type</th>
<th>N bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BOOL</td>
<td>Boolean</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>INT</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>REAL</td>
<td>Real numbers</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 11: Excerpt from IEC61131-3 Table 10 – Elementary data types

This section enumerates the parts of the product that should be modified to make new data types available to the user.

- In presentation layer:
  - Class “VariableDialogForm” -> ComboBox “Type” -> Property “Items”: a new item should be added.

- In domain layer:
  - In Generic/DataTypes folder:
    - A new subclass of “Variable” class should be defined.
    - Class “VariablesManager”: A case for the new data type should be added in several functions.
  - In Grafcet/ControllableGc/GCFunctions folder:
    - Class “GCFunctionsController” -> Function “GetMainH”: A case for the new data type should be added.
Design model

The design model is represented by means of the View of Participating Classes and the sequence diagram for each use case.

Aiming to be a valuable complement of the product source code, these diagrams do not reflect minor auxiliary operations, but only the main distribution of responsibilities during the use case.

New project
Save project
Open project
New grafcet
Add initial step
Add step

Same as in Add initial step.
Add project variable

Diagram showing the relationships between variables and forms in a project.
Add grafcet variable
Add transition
Code generation
Run

Enter run mode
Start run
Generate code
Stop run
Add action
Add macro-step

Diagram:

- BodyForm
- MacroStepDialogForm

- Body
- GraphicalStepsList

- GraphicalMacroStop
Edit macro-step

BodyForm ----> MacroStepDialogForm

:User
b:BodyForm

MacroStepDialogForm

DoubleClick
ShowDialog
ShowBody
ShowBody
msh:BodyForm

new
new
Show
Add enclosing Step
Edit enclosing step
Show enclosure
Add forcing order
Implementation

DRY improvements
Several “don't repeat yourself improvements” [WP:DRY] were made upon the original code of UniSim.

Snap
In the context of diagram drawing, the snap coefficient defines the separation between lines of a virtual grid. Symbols in the diagram can only be placed where the grid lines intersect.

In UniSim, the definition of snap coefficient (“CoeffSnap”) and a function that applies it to a given coordinate (“Snap”) were located in classes “FormSfc”, “GraphicalTransition”, “GraphicalTransitionsList”, and “Sfc”.

This was discovered when trying to change the snap for a larger one, since the original was considered to be too fine.

To avoid this duplicity, a new public class named "Drawer" was made, offering a public method "GetSnap", to get the snap value.

Small rectangles
Two small rectangles appear on top and bottom of all steps of a grafcet when a new transition is being placed.

Illustration 36: UniSim's small rectangles for steps.

Definition of the location and size of these rectangles was done in class "GraphicalStep" for drawing, but for mouse click detection it was done in class BaseGraphicalStep instead.

This was discovered when trying to change the size of these rectangles for a larger one, since the original was considered to be too small.

Illustration 37: Small rectangles for steps of the new system.
To avoid this duplicity, four new public methods "SmallRetX", "SmallRetY", "SmallRetLarg", "SmallRetAlt", were made on class BaseGraphicalStep.

**Implementation tradeoffs**

In the face of some implementation complexities that, together with schedule constraints, would have prevented the implementation of some parts of the product, it was prioritized to provide samples of the implementation of all the qualitatively different ones, rather than providing full implementations of some and no implementation at all of some others.

**Action graphics**

IEC60848 specifies different symbols for different types of actions.

The symbol for continuous actions without condition is a rectangle.

![Illustration 38: IEC60848 symbol for continuous action.](image)

The symbols for continuous actions with condition and stored actions have, additionally, some trait on top of that rectangle to represent the condition or the event associated to that action.

![Illustration 39: IEC60848 symbol for time-dependent continuous action.](image)

![Illustration 40: IEC60848 symbol for stored action on rising edge.](image)
In order to represent several actions per step, IEC60848 specifies different, equivalent, ways.

Illustration 41: IEC60848 different representations of the association of several actions at one step.

However, as of ed2.0, IEC60848 only exemplifies one way to represent multiple actions when one or more of them have some associated condition or event.

Illustration 42: Excerpt from IEC60848 ed2.0 annex B3.

In the other hand, the way UniSim represents several actions per step is piled vertically, such as in representation (4) of the illustration above.

Illustration 43: Several actions per step, as represented by UniSim.

UniSim is based on SFC instead of GRAFCET. In SFC, actions are not represented with particular graphical traits, but with text areas on the left and right of the “main” area - where the assignation or allocation is noted.

For this project, it was discarded to provide a representation close to the GRAFCET norm. Instead, it was reused what UniSim provided already.
From that point, only indispensable changes were made to represent GRAFCET actions, instead of SFC actions. These changes consisted in placing a wide text area to the left of the “main” area. This left area is used to note different things depending on the action type.

- For continuous actions: The assignation condition.
- For stored actions: The event that triggers the allocation.

**Illustration 44: Several actions per step, as represented by the product.**

**Actions code generation**

For this project, it was discarded to provide an implementation of images for analog variables, and automatic translation. Thus, only images for boolean variables were implemented. This shown also by the following table.

<table>
<thead>
<tr>
<th></th>
<th>Input image</th>
<th>Output image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean variables</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Analog variables</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Automatic translation</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**CRUDs**

CRUD stands for Create, Read, Update, and Delete.

For this project, the implementation of creation and read use cases was prioritized over update and delete use cases.

This allowed to focus on developing a structurally complete prototype, rather than on a more usable, but structurally incomplete one.

As a result of this prioritization, some update and delete use cases did not fall within time for implementation, but all create and read use cases did.
**Economic assessment**

**Material resources cost**

**Hardware**
The Spanish Real Decreto 1777/2004 defines an amortization period of 3 years for a computer. Assuming this amortization rate and disregarding network hardware and printers because they were also used by more than 50 people, hardware costs are listed in the following table.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Quantity</th>
<th>Acquisition cost (€)</th>
<th>Attributable cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003 desktop PC</td>
<td>1</td>
<td>600</td>
<td>0</td>
</tr>
<tr>
<td><strong>Attributable total</strong></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

**Software**
The Spanish Real Decreto 1777/2004 defines an amortization period of 3 years for computer software. Assuming this amortization period and a usage period of 6 months, software costs are listed in the following table.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Quantity</th>
<th>Acquisition cost (€)</th>
<th>Attributable cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows XP</td>
<td>1</td>
<td>250</td>
<td>0</td>
</tr>
<tr>
<td>Professional</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visual Studio 2008</td>
<td>1</td>
<td>300</td>
<td>50</td>
</tr>
<tr>
<td>BOUML</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Attributable total</strong></td>
<td></td>
<td></td>
<td>50</td>
</tr>
</tbody>
</table>

**Buildings**
Disregarding the cost of using the CITCEA premises, the cost of the project in buildings is zero.

**Other services**
Assuming a price of 0.15€ per kWh for the electricity service, and disregarding the cost of using the CITCEA access to the Internet, the cost of the project in other services is listed in the following table.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Weekly cost (€)</th>
<th>Period in weeks</th>
<th>Total price (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electricity</td>
<td>4</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
Waste management

Only recyclable paper was used for this project. The cost of this paper recycling is zero.

Total

<table>
<thead>
<tr>
<th>Concept</th>
<th>Cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>0</td>
</tr>
<tr>
<td>Software</td>
<td>50</td>
</tr>
<tr>
<td>Buildings</td>
<td>0</td>
</tr>
<tr>
<td>Other services</td>
<td>100</td>
</tr>
<tr>
<td>Waste management</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>150</td>
</tr>
</tbody>
</table>

Human resources cost

Assuming salaries listed in “Estudio Remuneración 2011 Tecnología” published by Michael Page human resources consultancy, plus a 65% in Social Security taxes, the cost of the project in human resources is listed in the following table.

<table>
<thead>
<tr>
<th>Role</th>
<th>Price/time (€/h)</th>
<th>Time (h)</th>
<th>Price (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS manager</td>
<td>52</td>
<td>20</td>
<td>1040</td>
</tr>
<tr>
<td>Development manager</td>
<td>36</td>
<td>20</td>
<td>720</td>
</tr>
<tr>
<td>Project manager</td>
<td>32</td>
<td>50</td>
<td>1600</td>
</tr>
<tr>
<td>Analyst</td>
<td>17</td>
<td>160</td>
<td>2720</td>
</tr>
<tr>
<td>Designer</td>
<td>48</td>
<td>300</td>
<td>14400</td>
</tr>
<tr>
<td>Programmer</td>
<td>24</td>
<td>100</td>
<td>2400</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>650</td>
<td>22880</td>
</tr>
</tbody>
</table>

Total cost

<table>
<thead>
<tr>
<th>Concept</th>
<th>Cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material resources</td>
<td>150</td>
</tr>
<tr>
<td>Human resources</td>
<td>22880</td>
</tr>
<tr>
<td>Total</td>
<td>23030</td>
</tr>
</tbody>
</table>
Conclusions

Achieved goals
It has been specified and designed a software system that allows to:

• Edit GRAFCET diagrams.
• Verify GRAFCET diagrams.
• Generate code in C language that has a behavior specified by a GRAFCET diagram, with the property that this code adheres to the GeCé methodology.
• Store and load GRAFCET diagrams in disk.

Future work
The following improvements can be made over the system:

• In the diagram editor:
  ◦ Pressing the enter key confirms the addition of a transition.
  ◦ Arithmetic comparisons don't need to be enclosed between “{” and “}”.
  ◦ Zoom for the drawing area.
  ◦ Variables can be declared when they are used for the first time.
  ◦ Device profiles can be defined. Consisting in dictionaries of variables and their mapping into the industrial control system (inputs, outputs, …).
  ◦ New data types can be defined by the user, in runtime.
  ◦ For clarity, upward connections between steps are represented with numeric references, rather than with upward lines.
  ◦ For clarity, connections between steps are optionally represented with numeric references, instead of lines.

• In the code generator:
  ◦ Input and output images for analog variables.
  ◦ Automatic translation between variables and the images of those variables.

• In the persistence layer:
  ◦ Find and use an standard GRAFCET markup language to store and load diagrams interoperable with other editors.
Bibliography

- GPL - GNU General Public License [on line] [Retrieved: 2011-01-07] Available at: http://www.gnu.org/licenses/gpl.html
- JGRAFCHART - The Grafchart Homepage [on line] [Retrieved: 2011-01-13] Available at: http://www3.control.lth.se/user/grafchart/
- PLCOPEN - PLCopen Home [on line] [Retrieved: 2011-01-12] Available at:
http://www.plcopen.org/


- TELERIK - Convert VB to C# or C# to VB [on line] [Retrieved: 2011-08-28] Available at: http://converter.telerik.com/

- UNISIM - UniSim [on line] [Retrieved: 2011-01-07] Available at: http://wpage.unina.it/detommas/unisim/


- VOLERE - Volere Requirements Specification Template [online] [Retrieved: 2011-02-02] Available at: http://www.volere.co.uk/template.htm


Annex A: Application example

Following there is a simplified example of application of the product developed during this project. Summary of steps:

1. System specification
2. Diagram drawing
3. Diagram validation
4. Code generation
5. Code compilation
6. Code execution

System specification

We want to automate the control of a garage door. We want the following behavior for the door.

- The door is initially closed
- The door will open when the "open" command is issued
- 10 seconds after being opened, the door will close
- If during closing, an obstacle is detected, the door will open again

Diagram drawing

Using the product, we start a new project, add a new grafcet, introduce the variables and draw the diagram.

Illustration 45: Project menu of the product, before creating a project.
Illustration 46: Project menu of the product, after creating a project.

Illustration 47: Diagram edition interface of the product.
Diagram validation

Then we proceed to validate the behavior of the diagram.

Illustration 48: Run command in the project menu.

Illustration 49: Diagram validation interface of the product.
Code generation

Then the code is generated. A number of source and header files is produced.

Illustration 50: Code generation option in the project menu.

Illustration 51: Source files generated by the product.
**Code compilation**

The generated code is then compiled by the proper tool, such as Microchip MPLAB.

[Image of Microchip MPLAB IDE]

*Illustration 52: Microchip MPLAB IDE.*

**Code execution**

Finally, the code is ready to be transferred to the microcontroller and executed by it, providing the desired behavior for the garage door.

[Image of garage door control circuitry]

*Illustration 53: Garage door while being opened.*

[Image of garage door control circuitry]

*Illustration 54: Garage door control circuitry.*
Annex B: Guidelines for the extension of GeGé

The following guidelines are aimed at new developers of the system. They are not axioms, but heuristic knowledge gained while trying to use a combination of Scrum and RUP during my work on the system, which includes, remarkably, the extension of a part of UniSim (the IEC1131-3 editor).

User story is the measure of all things

From Wikipedia:

In computer programming a user story is one or more sentences in the everyday or business language of the end user that captures what the user wants to achieve.

Don't approach your work as “functions that have to be written”, but as “user stories that have to be implemented”.

Functions, parameters, attributes, etc. are nothing but means to implement user stories.

For instance: don't approach your work as “creating a function that parses temporary conditions together with Boolean expressions”, but as “implementing a user story where the user sets a transition with a time-dependent condition”.

This will allow you to focus on user value, rather than on technical obstacles, giving your daily work a higher sense of usefulness, and a clearer vision of what is really necessary and what can be simplified.

To be able to implement any given user story, it is enough to understand how similar user stories are implemented.

To understand how a user story is implemented, it is enough to debug step by step how it is resolved.

One user story at a time

From some smart guy:

The best way to do a lot of things is one by one.

User stories should be implemented one by one.

Working on several user stories at a time, at any layer (presentation, domain, infrastructure, ...), can easily result in additional implementation delay due to the extra number of known and unknown variables at stake.

When in Rome, do as the Romans

For starting, use the design patterns used in the code as is.

To learn them, trace the implementation of use cases with similarities with the one you are working on. Do your own sequence diagrams as you find it useful.

It is better to solve each problem with one bad design pattern, than solving it with two or more design patterns (no matter how good they are). The value of a design pattern resides in the “pattern” word itself. The more patterns, the less pattern; the less simplicity; the less predictability; the more chaos.
Only after identifying and learning which are the existent design patterns, you might replace them.

**Use the analysis class diagram**

Rather than the design class diagram.

The design class diagram of the whole system is too big to be useful.

The design class diagram, nevertheless, is the result of applying the design patterns to the analysis class diagram.

Again, it is important to have a small number of design patterns used for each problem, because once you know the analysis diagram, you can make a better guess on the design diagram. Best is one and only one design pattern for each problem.

**The Tetris law**

Assign responsibilities as you should (based on the analysis class diagram and the design patterns), not simply as you can.

Poorly assigned responsibilities are like poorly placed pieces on a Tetris game; if you accumulate too many, the game is over.

As sometimes is nearly impossible to know what will stand as a good responsibility assignation:

1. Don't get paralyzed by analysis; do a first assignation moderately well.
2. As it becomes clear how the code should be refactored, be brave and refactor it with no delay.

**The mitosis law**

Cells don't grow indefinitely, but they perform mitosis.

As more user stories are implemented, don't grow functions and classes indefinitely; divide them as many times as needed to balance complexity equally among them.

**Code is not free**

Maintaining a program with an infinite number of lines of code is infinitely hard.

Add code only if it provides more value than what it costs.

User needs evolve. Remove code when it is providing less value than what it costs.

**Corollary: The GeGé extension recipe**

To extend GeGé, don't think about extending its code, but about extending its user stories.

1st) Understand clearly what you want the new user story to be.
2nd) Extend the analysis class diagram as it *should* be extended.
3rd) Trace some similar user story step by step, and get its design patterns.
4th) Apply changes to the code. These changes will be the combination of changes over the analysis class diagram plus the corresponding design patterns.
5th) Celebrate. You just gave value to users.
Annex C: Sprint meetings log

Sprint 1

Sprint planning
Day: 2011-02-14
Start/End hour: 16:40/17:10
Room: Mecatrònica planta 2
Attendees:
- Samuel Gómez (Team)
- José Pedro Muñoz (Product Owner)
Planned features:
- Edition: Step edition
- Edition: Transition edition
- Compilation: Step compilation
- Compilation: Transition compilation

Sprint review
Day: 2011-02-16
Start/End hour: 12:00/12:10
Room: Mecatrònica planta 2
Attendees:
- Samuel Gómez (Team)
- José Pedro Muñoz (Product Owner, User)
- Three additional spontaneous onlookers, including at least one User.
Demo:
- The planned features were successfully demoed.
Feedback gathered:
- All demoed features were validated.
- All stakeholders seemed satisfied.
Sprint 2

Sprint planning
Day: 2011-02-16
Start/End hour: 12:10/12:20
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • José Pedro Muñoz (Product Owner)
Planned features:
  • Structuring: Multiple partial grafcets

Sprint review
Day: 2011-02-18
Start/End hour: 18:20/18:30
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • José Pedro Muñoz (Product Owner, User)
Demo:
  • The planned features were successfully demoed.
Feedback gathered:
  • Partial grafcets should be renamed to “grafcets”.
  • Bug #1 was reported.
  • The product owner seemed happy.


**Sprint 3**

**Sprint planning**

Day: 2011-02-22  
Start/End hour: 17:00/17:30  
Room: Mecatrònica planta 2  
Attendees:  
- Samuel Gómez (Team)  
- José Pedro Muñoz (Product Owner)

Planed features:  
- Transition with condition: Boolean predicate

**Sprint review**

Day: 2011-03-01  
Start/End hour: 15:00/15:05  
Room: Mecatrònica planta 2  
Attendees:  
- Samuel Gómez (Team)  
- José Pedro Muñoz (Product Owner, User)

Demo:  
- The demo of the planned features triggered multiple feedback.

Feedback gathered:  
- The value of the variables should be visible in the variable selector and editor.  
- Two different scopes should be possible for variables: project and grafcet.  
- Variables should be modifiable (so far they can only be added and deleted)  
- The value of integer variables should be editable in hexadecimal.  
- The user has encountered difficulties to click on transitions since they are quite small.
Sprint 3.1

Sprint planning
Day: 2011-03-01
Start/End hour: 15:05/15:10
Room: Mecatrònica planta 2
Attendees:
• Samuel Gómez (Team)
• José Pedro Muñoz (Product Owner)
Planned features:
• Variable editor and selector will show the variable value.
• Variables will have two possible scopes: project and grafcet.
• Variables will be modifiable.
• Value for integer variables will be specifiable in hexadecimal.
• Larger clickable area for transitions.

Sprint review
Day: 2011-03-03
Start/End hour: 15:20/15:30
Room: Mecatrònica planta 2
Attendees:
• Samuel Gómez (Team)
• José Pedro Muñoz (Product Owner, User)
Demo:
• The planned features were successfully demoed.
• Other non planed features where maliciously tested by the product owner, leading to application crashes.
Feedback gathered:
• All demoed features were validated.
Sprint 4

*Sprint planning*
Day: 2011-03-08
Start/End hour: 12:45/12:55
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • José Pedro Muñoz (Product Owner, User)
Planned features:
  • Transition with condition: time after rising edge of step state.

*Sprint review*
Day: 2011-03-10
Start/End hour: 16:20/16:25
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • José Pedro Muñoz (Product Owner, User)
Demo:
  • The planned features were successfully demoed.
Feedback gathered:
  • All demoed features were validated.
Sprint 5

**Sprint planning**
Day: 2011-03-10
Start/End hour: 16:25/16:50
Room: Mecatrònica planta 2
Attendees:
- Samuel Gómez (Team)
- José Pedro Muñoz (Product Owner, User)
- Guillem Coromina (CITCEA employee)

Planned features:
- Verification
  - Compilation of the generated code.
  - Execution of the generated code
  - Communication with the executed code.
    - Reading values from code execution.
    - Writing values to code execution.

**Sprint review**
Day: 2011-03-29
Start/End hour: 12:00/12:10
Room: Mecatrònica planta 2
Attendees:
- Samuel Gómez (Team)
- José Pedro Muñoz (Product Owner, User)

Demo:
- The planned features were successfully demoed.

Feedback gathered:
- Pressing enter should confirm the addition of a transition.
Sprint 6

**Sprint planning**
Day: 2011-03-29
Start/End hour: 12:10/12:20
Room: Mecatrònica planta 2
Attendees:
- Samuel Gómez (Team)
- José Pedro Muñoz (Product Owner, User)
Planned features:
- Sequencing
  - Activation of parallel sequences
  - Synchronization and activation of parallel sequences
- Pressing enter confirms the addition of a transition

**Sprint review**
Day: 2011-03-04
Start/End hour: 13:10/13:30
Room: Mecatrònica planta 2
Attendees:
- Samuel Gómez (Team)
- José Pedro Muñoz (Product Owner, User)
Demo:
- The planned features were successfully demoed.
Feedback gathered:
- “false” should be a possible value for transition condition.
- During a run, grafcet elements should not show their selection state.
- Ending a running should can be done through the program, not through the operating system's task administrator.
- After a run, steps should not show their activation state.
**Sprint 7**

**Sprint planning**

Day: 2011-04-04  
Start/End hour: 13:30/16:00  
Room: Mecatrònica planta 2  
Attendees:  
- Samuel Gómez (Team)  
- José Pedro Muñoz (Product Owner, User)  
Planned features:  
- Run control: Set variable values.  
- “false” will be a condition for transitions.  
- During a run, grafcet elements will not show their selection state.  
- Ending a running will be able to be done through the program.  
- After a running, steps will not show their activation state.

**Sprint review**

Day: 2011-05-02  
Start/End hour: 15:05/15:15  
Room: Mecatrònica planta 2  
Attendees:  
- Samuel Gómez (Team)  
- José Pedro Muñoz (Product Owner, User)  
Demo:  
- The planned features were successfully demoed.  
Feedback gathered:  
- Icon “Set step as initial” can confuse the user. It is understood as “Add initial step”.  
- Bug discovered: The “step” button has to be pressed several times to allow the grafcet evolve.  
- Variables' forms should allow the current values of the variables values to be edited by clicking or double clicking on the value itself, rather than through the use of a form button.
Sprint 7.1

Sprint planning
Day: 2011-05-02
Start/End hour: 15:15/15:20
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • José Pedro Muñoz (Product Owner, User)
Planned features:
  • Use the icon of the feature “Set step as initial” for a new feature named “Add initial step” and remove the former feature.
  • Fix bug: The “step” button has to be pressed several times to allow the grafcet evolve.
  • Variables' forms will allow the current values of the variables to be edited by clicking or double clicking on the value itself, rather than through the use of a form button.

Sprint review
Day: 2011-05-04
Start/End hour: 13:15/13:25
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • José Pedro Muñoz (Product Owner, User)
Demo:
  • The planned features were successfully demoed.
Feedback gathered:
  • Arithmetic comparisons should not need to be enclosed between “{“ and “}”
Sprint 8

Sprint planning
Day: 2011-05-04
Start/End hour: 13:25/14:00
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • José Pedro Muñoz (Product Owner, User)

Planned features:
  • Continuous actions without condition.

Sprint review
Day: 2011-05-09
Start/End hour: 10:30/12:15
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • José Pedro Muñoz (Product Owner, User, Director)

Demo:
  • A partial implementation of continuous actions without condition is demoed. In particular, the part not implemented is the mapping between output variables of both domains (graphic editor and generated code). The cause is that further requirements have to be extracted.

Feedback gathered:
  • The development methodology followed so far is strongly discussed. Product Owner, and User, who turns to be also the director of the project, requests for a more predictive development process.
    ◦ The process followed so far was:
      ▪ Features are defined at a high level at the beginning.
      ▪ Close definition of features, definition of presentation, and analysis, design and implementation tasks are performed iteratively.
    ◦ The process set to be followed from now on is:
      ▪ Excepting special cases, features and presentation should be totally defined at the beginning.
      ▪ Iterations will not contain definition of features and presentation, but only analysis, design and implementation tasks.
- Actions should be definable in C code, but abstracted from the names of GeCé's internal variables.
Sprint 9

Sprint planning
Day: 2011-05-09
Start/End hour: 12:15/12:30
Room: Mecatrònica planta 2
Attendees:
• Samuel Gómez (Team)
• José Pedro Muñoz (Product Owner, User)
Planned tasks:
• Close definition of remaining features.
• Mockups for remaining presentation items.

Sprint review
Day: 2011-05-12
Start/End hour: 17:30/17:33
Room: Mecatrònica planta 2
Attendees:
• Samuel Gómez (Team)
• José Pedro Muñoz (Product Owner, User, Director)
Demo:
• Planned demo consists in showing proposals for:
  ◦ Enclosed step form.
  ◦ Enclosing step form.
  ◦ Forcing order form.
  ◦ Visual representation of a large number of actions per step.
  ◦ A solution to allow the user to program actions in C code, but abstracted from GeCé's internal variables names, and an estimation of its development cost.
  ◦ Listing the existent macro-steps and enclosing steps in the project form.
Demo events:
• Product Owner, User, and Director (JPM) refuses to assess proposals individually, and asks the team whether they think that that proposal makes a really good grafcet diagrams editor.
• The team responds that it depends on the definition of good. JPM responses asking whether the team thinks that that proposal makes a great grafcet diagrams editor.
• The team responds that that depends on the expectations of the users. JPM responses asking
whether the team thinks that that proposal makes the best grafcet editor of the market.

- The team answers “yes”, euphorically. JPM says “So do it and do it on the time you said you'll do it.”, referring to the development time estimation made 4 weeks ago.
Sprint 10

Sprint planning
Day: 2011-05-12
Start/End hour: 18:00/18:30
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
Planned features:
  • Continuous actions without condition
    ◦ Including distinction between Input and Output variables

Sprint review
Day: 2011-05-13
Start/End hour: 14:15/14:25
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • Enric Agüera (User)
Demo:
  • The planned features were successfully demoed.
Feedback gathered:
  • Action Dialog Form should show only output variables
  • Action Dialog Form font size should be bigger
  • Double click on Action Dialog Form should submit the form
  • Grafcet and project variables tables should show a column indicating the application of the variable (input, internal, output)
  • Boolean input variables should be settable faster and using only the mouse
  • Exit from the Run Control Form should stop the running
**Sprint 10.1**

**Sprint planning**

Day: 2011-05-13  
Start/End hour: 14:25/14:30  
Room: Mecatrònica planta 2  
Attendees:  
• Samuel Gómez (Team)  
Planning features:  
• Action Dialog Form will show only output variables  
• Action Dialog Form font size will be bigger  
• Double click on Action Dialog Form will submit the form  
• Grafcet and project variables tables will show a column indicating the application of the variable (input, internal, output)  
• Boolean input variables will be settable faster and using only the mouse  
• Exit from the Run Control Form will stop the running

**Sprint review**

Day: 2011-05-13  
Start/End hour: 17:40/17:50  
Room: Mecatrònica planta 2  
Attendees:  
• Samuel Gómez (Team)  
• Enric Agüera (User)  
Demo:  
• The planned features were successfully demoed.  
Feedback gathered:  
• The diagram should be zoomable.  
• Variables should be declarable on the fly; as they are used without being declared before.  
• Grafcet form should show also project variables; not only grafcet variables. And a new column showing the scope of the variable (project or grafcet).  
• Scope of variables should be changeable.
Sprint 10.2

Sprint planning
Day: 2011-05-13
Start/End hour: 17:50/18:00
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
Planned features:
  • Grafcet form will show also project variables; not only grafcet variables. And a new column showing the scope of the variable (project or grafcet).
  • Scope of variables will be changeable.

Sprint review
Day: 2011-05-31
Start/End hour: 12:30/12:35
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • Enric Agüera (User)
Demo:
  • The planned features were successfully demoed.
Feedback gathered:
  • User is pleased and remarks usefulness of that features.
Sprint 11

Sprint planning
Day: 2011-05-17
Start/End hour: 11:10/11:15
Room: Mecatònica planta 2
Attendees:
  • Samuel Gómez (Team)
Planned features:
  • Stored action: on rising/falling edge of step variable

Extraordinary meeting
Day: 2011-05-18
Start/End hour: 17:30/18:00
Room: Mecatònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • José Pedro Muñoz (Product Owner, User)
  • Joan Bergas (User)
Feedback gathered:
  • As a future feature: Device profiles should be definable. They would consist in dictionaries of variables and their mapping into the industrial system (inputs, outputs, …).
  • As a future feature: For clarity, upward connections between steps should be represented with numeric references, rather than with upward lines.
  • As a future feature: For clarity, connections between steps should be optionally represented with numeric references, instead of lines.
  • User comments indicate different degrees of value for some of the not yet implemented features.
    ◦ The following features would have low value:
      ▪ Stored action on transition clearing
      ▪ Stored action on event
    ◦ The following features would have high value:
      ▪ Continuous action with time-dependent condition
      ▪ Transition on time after falling edge
      ▪ Forcing order
**Sprint review**

Day: 2011-05-31  
Start/End hour: 12:35/12:45  
Room: Mecatrònica planta 2  
Attendees:  
- Samuel Gómez (Team)  
- Enric Agüera (User)  

Demo:  
- A partial implementation of the planned features is successfully demoed. Translation from names of variables to names of images (see chapter “Design”) is not yet implemented.

Feedback gathered:  
- Bug discovered: Several variables with the same name can be declared in different scopes, making the code generation to crash during variable mapping.
Sprint 12

Sprint planning
Day: 2011-05-23
Start/End hour: 11:10/11:15
Room: Mecatrònica planta 2
Attendees:
- Samuel Gómez (Team)
Planned features:
- Macro-steps

Sprint review
Day: 2011-05-30
Start/End hour: 19:00/19:02
Room: Mecatrònica planta 2
Attendees:
- Samuel Gómez (Team)
- José Pedro Muñoz (Product Owner, User)
Demo:
- The planned features were successfully demoed
Feedback gathered:
- None
Sprint 13

**Sprint planning**
Day: 2011-05-30
Start/End hour: 19:08/19:10
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
Planned features:
  • Forcing orders

**Sprint review**
Day: 2011-06-22
Start/End hour: 16:20/16:25
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • Enric Agüera (User)
Demo:
  • The planned features were successfully demoed.
Feedback gathered:
  • None.
**Sprint 14**

**Sprint planning**
Day: 2011-06-01  
Start/End hour: 19:08/19:10  
Room: Mecatrònica planta 2  
Attendees:  
• Samuel Gómez (Team)  
Planned features:  
• Enclosing steps

**Sprint review**
Day: 2011-06-22  
Start/End hour:  
Room: Mecatrònica planta 2  
Attendees:  
• Samuel Gómez (Team)  
• Enric Agüera (User)  
Feedback gathered:  
• Bug discovered: The enclosing step to which an enclosing step form belongs is not visible.  
• Bug discovered: Two grafcets with the same name can be declared.
Sprint 15

Sprint planning
Day: 2011-06-07
Start/End hour: 14:30/14:40
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
Planned features:
  • Persistence: Save grafcets to disk
  • Persistence: Load grafcets from disk.

Sprint review
Day: 2011-06-14
Start/End hour: 18:50/18:55
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  • José Pedro Muñoz (Product Owner, User, Director)
Feedback gathered:
  • The team explains that design, proof of technology, and an initial implementation for saving and loading are complete. However, a full implementation will require one week. The team asks for skipping full implementation until other remaining important features are developed.
  • The director agrees to skip full implementation of saving and load if the schedule for other features is at stake.
  • Since it is, the next sprint is started.
Sprint 16

Sprint planning
Day: 2011-06-15
Start/End hour: 12:00/12:05
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
Planned features:
  • Continuous action with condition: time dependent assignment condition.

Sprint review
Day: 2011-06
Start/End hour:
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
  •
Demo:
  •
Feedback gathered:
  •
Sprint 17

Sprint planning
Day: 2011-06-17
Start/End hour: 18:00/18:05
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
Planned features:
  • Transition on time after falling edge

Sprint review
Day: 2011-06-
Start/End hour:
Room: Mecatrònica planta 2
Attendees:
  • Samuel Gómez (Team)
Team notes:
  • As a result of the integration of action conditions and transition conditions, the unplanned features “Continuous action with condition: Boolean predicate”, and “Transition with condition: time-dependent transition” resulted to have technical precedence regarding the planned feature “Transition on time after falling edge”. So those were also implemented during this sprint.

Demo:
  •

Feedback gathered:
  •
Future sprints

Feedback requests not satisfied so far:

- Pressing enter will confirm the addition of a transition.
- Arithmetic comparisons should not need to be enclosed between "{" and "}".
- Actions should be definable in C code, but abstracted from the names of GeCé's internal variables.
- The diagram should be zoomable.
- Variables should be declarable on the fly; as they are used without being declared before.
- As a future feature: Device profiles should be definable. They would consist in dictionaries of variables and their mapping into the industrial system (inputs, outputs, …).
- As a future feature: For clarity, upward connections between steps should be represented with numeric references, rather than with upward lines.
- As a future feature: For clarity, connections between steps should be optionally represented with numeric references, instead of lines.