MASTER IN COMPUTER SCIENCE

Improvement of FDNet human interface

Delange, Fabien; Purnelle, Paul-Emile

Award date:
2005

Facultés Universitaires Notre-Dame de la Paix
Institut d'informatique
Rue Grandgagnage 21, 5000 NAMUR
Tél. 081-72 50 02 • fax 081-72 49 67

Improvement of FDNet Human Interface

Fabien Delange
Paul-Emile Purnelle

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Facultés Universitaires Notre-Dame de la Paix
Institut d'informatique
Rue Grandgagnage 21, 5000 NAMUR
Tél. 081-72 50 02 • fax 081-72 49 67

Improvement of FDNet Human Interface

Prof. Pierre-Yves SCHOBBENS,
Prof. Satoshi TADOKORO

Fabien Delange
Paul-Emile Purnelle

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Résumé

Ce mémoire à été réalise dans le cadre d'un stage dans un laboratoire de robotique situé à Kobe. Le projet auquel nous avons participé est orienté « aide au sauvetage » de victimes d'incidents naturels. Nous étions intégré dans une équipe d'ingénieurs électroniciens chargés de l'élaboration de robots semi autonomes. Ces robots sont dirigés par un même programme et travaillent en parallèle.

Nos objectifs consistaient à améliorer une interface graphique permettant de gérer la couche décisionnelle d'un groupe de robots.

Nous commencerons donc par introduire le contexte d'élaboration de FDNet, la couche interessée. Ensuite, nous tenterons de positionner le réseau par rapport aux concurrents existants. C'est alors que nous décrirons l'Interface Homme Machine avant de développer les améliorations demandées par le laboratoire. Nous proposerons enfin nos solutions et les critiquerons afin de fournir aux futurs développeurs des bases propices.

Abstract

This thesis has been written in the context of a work experience in a robotics' lab situated in Kobe, Japan.

The project we took part of was meant to help rescuers of natural disaster's victims.
We joined a team of engineers specialized in electronic in charge of the elaboration of half autonomous robots. Those robots are under control of one unique software and work all together.

Our aim was to improve the graphic interface allowing to manage the decisional layer of a robot's pool.

We will start to introduce the elaboration's context of FDNet, the layer we are talking about. Then, we will try to set the network towards existing rival. We will then describe the HI before developing improvements required by the lab.

Eventually, we will propose our solutions and will criticism each of them in order to provide to future researchers favorable bases.

Acknowledgment

We want to thank Mr. P.-Y. Schobbens for his help and his support while the project and the writing of this thesis.

We also want to thank Mr. Tadokoro for his welcome and his support. As well we want to thank all the IRSI members for their help for all the administrative work and for kindly helped us to fit into Japan.

One more time, thank you very much all!

Table of content

I. Introduction

The International Rescue System Institute *(IRS)* [1] *is the industry-government-academia-civilian research organization to advance and diffuse high-technologies coping with disaster.*

Its aim? Contribute to build a safe society based on various organization and human resources which would enable people to live in confidence in their society.

It has been ten years since the Great Hanshin Awaji Earthquake hit a Japanese urban area and took 4,571 lives in Kobe city alone. Technological development for disaster reduction has reached various stages. One of them is a rescue robot capable of searching for victims trapped under the wreckage of collapsed buildings. IRS is taking a core role in "the disaster countermeasure research" of the "Special Project for Earthquake Disaster Mitigation in Urban Areas". This project is launched by MEXT* and *aims at significant mitigation the earthquake disaster damage on the scale of the Great Hanshin Earthquake, in big city regions such as Tokyo metropolitan area and Keihanshin area*. Another aim of this research project is also *to establish a basis of science and technologies for earthquake disaster-prevention measures.*

Research and development of robots, intelligent sensors, portable devices and human interfaces which could integrate and gauge the information in disaster confusion and make the best rescue efforts suited to the situation is part of the work of IRS. *IRS aims to go a long way toward lifesaving or other human activities and decision-making in disaster response by advancing the research and development for active and intelligent data acquisition and integration on the network.*

II. FDNet and the RoQ team

A. FDNet, a Flat Distributed Network Architecture

Based in Japan, *the International Rescue System Institute* works on the robot fields and more especially on rescue robots. The first issue of their work is to improve the particularities of rescue robots to make them able to cope the best with the complexity of their work field. Actually rescue robots have to operate in difficult and complicated places, helping rescuers to find victims in destroyed environments, after an earthquake or a disaster for example. Because of the damages caused by each disaster, every situation is different and unexpected complication can always occur

* MEXT: Ministry of Education, Culture, Sports, Science and Technology

while the robots work among the debris. That makes the work of the rescue robots fair arduous because topography of the intervention place is every time different and not entirely foreseeable.

The conception of a complete robot behavior is impossible because it is not possible to prepare the robots to fit every kind of environment, one after the other. Consequently, robots must have some form of intelligence: *a software architecture that combines the different basic and fundamental technologies and new skills learnt in that specific place*. This architecture could be FDNet, a Flat Distributed Network Architecture.

Many previous studies have been made about robot architecture, but because none of them had structures to perform dynamic self-organization or dynamic reconfiguration, they could not been used for the rescue robots. IRSI researchers could have tried to modify these architectures, but they chose to create a specific architecture common for all rescue robots by taking different characteristics from each of them. That's how originate FDNet, which is based on three previous architectures: ORiN, ORCA and Open-R.
Like its bases, *FDNet is a flexible, extensible and generic architecture.*

1. Flexibility [2]

Because of the complexity and the differences between environments, a rescue robot can't be totally autonomous. Under these conditions, the robot needs to be ordered and monitored by a human operator. But, it isn't realistic to think of a human always giving detailed movement orders to the robot. It is necessary for the robot to be half autonomous. By half-autonomy, we imply that the robot must be able to perform advanced tasks given simple orders, but to wait for specific ones when it falls under conditions where human judgment becomes necessary.

This is the reason why a common software architecture that shows flexibility regarding both software and hardware is needed, the goal being to be able to describe an intelligent system with that architecture.

To build this flexibility, the system must be able to do three things:

1. First, it has to be based on basic behaviors, not too simple to bear a minimal meaning but not too complicated to be easily ordered.
2. Secondly, the system must provide an intelligent ordering system to be able to construct advanced behaviors from the simpler ones.
3. Finally, the system must be able to create new behaviors according to the environment in which it is executed. Being able to learn from its own work will greatly improve system's performances.

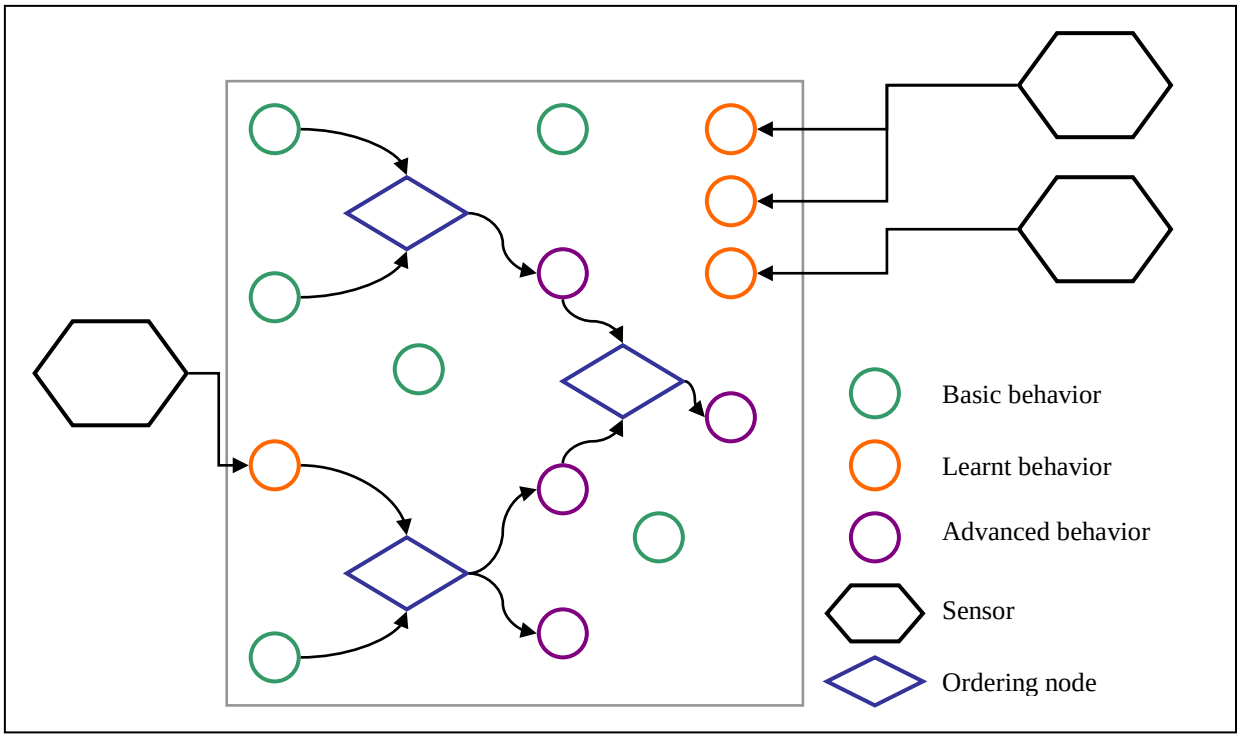


Figure 1 - FDNetworks' flexibility property

By creating advanced behaviors, based upon both basic and learnt ones, the robot will be able to find answers to problems specifically encountered on the field. Robot's reactions will then be exactly fitted for the actual environment he is maneuvering in.

2. Extensibility [2]

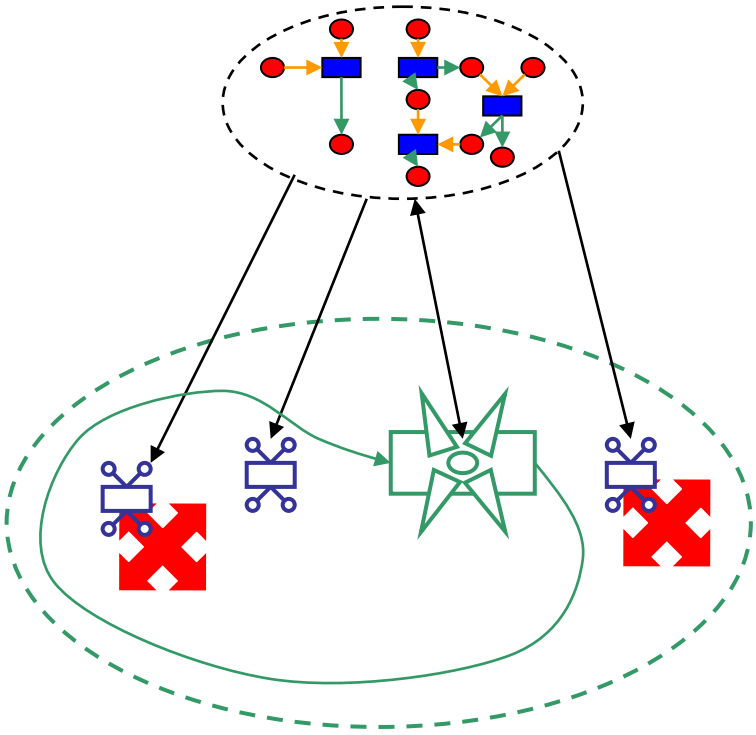
FDNet is created by using various layers. By dividing the architecture in different layers, it is easy to limit the impacts of future changes or improvements to the layer concerned by theses changes. This extensibility allows, for example, equipping robots with the most suitable sensors anytime, changing them as the environment change.

3. Generic architecture [2]

I.R.S.I researchers wanted to create a common architecture usable for all rescue robots whatever their type, with a common protocol allowing them to exchange data. This is FDNet's most important feature.

Rescue robots' ultimate goal is to help rescue injured people. Therefore, the best way to ensure that victims can be found and saved has to be put in place. Finding victims means to be able to get the best and the most complete information about the environment and to be able to discover the injured people using this information.

Various proposals have been made so far. For example, creating a group of robots consisting of "crawler robots", "legged robots" and "flying robots" has been thought of a lot. The flying robots provide general information about the environment and supervise crawlers and the legged robots, whose more specific researches will ensure victims can be found.



1: The **Helicopter** Robot is deployed on the devastated scene and will flight over the environment to provide information to the network.

2: Based on the information received from the Helicopter robot, the FDNetwork will give it orders to make general researches after casualties.

3: When something interesting is found, the FDNetwork sends the **Crawler** robots inside the **debris** to make more specific research for human lives.

Figure 2 - An example of cooperation between different kinds of robots

However, it is currently difficult to exchange information between these robots because each one has its own protocols and architectures. If all of them where using the same common protocol and the same architecture, not only would it be possible to make each robot able to discuss with each other, but it would also be possible to reduce their creation time, lots of complex problems encountered by a team being already resolved by other ones.

B. FDNet in more details [2]

Now that you have been introduced to FDNet's purpose, concept and origin, we will present each of its particularities in more details, basing ourselves on FDNet's definition.

1. FDNet is a Flat Distributed Network architecture

By Network, IRSI Researchers imply that an information network is used to create robot's intelligence. The robots having to be half autonomous, they have to be able to take some decisions like, for example, determining which direction is better to reach injured persons in a fragile terrain. To represent this intelligence, a neural-like network is used.

FDNetworks are made of two main components: the **Nodes** and the **Connections**. While Nodes consist of either raw information or processing objects, Connections are to be viewed as links between the Nodes, allowing the processing objects to access the information they need.

There are two kinds of Nodes:

1. **Data Node[1]:** This kind of Node represents Network's raw information, which can either arise directly from the Base Network itself or from the environment robots evolve in, by using sensors and cameras. New Data can also be processed by using Relation Nodes.
   All information contained in the Network is considered to be a feature. In other words, any data's value is decided in the same way: the device-level feature is decided in the same way that the system-level feature is.
2. **Relation node[2]:** Bears the same meaning in FDNet than the neuron specified in the recognition model or in neural networks. By using input Data, Relation nodes can also calculate new Data's value. In this case, the Relations will work with the help of "servants": agents having specific functions. Relations can only calculate the value of directly linked Data. But through the use of servants, they can access FDNetwork's whole structure. This way, Relations can perform Network's dynamic self-organization / reconfiguration.

Relations and Datas can have parameters. These parameters can be used to initialize a Node and to influence its behavior. Note that parameters are only represented as strings in the FDNet architecture. It means that they have no type, and don't bear any meaning by themselves. The Nodes using them will give them their meaning.

[1] To refer to Data Nodes, the term « Data » will also be used.
[2] To refer to Relation Nodes, the term « Relation » will also be used.

Relation and Data Nodes are linked by Connections, which can be of two types:

- Reader Connection: A Connection between a Data and a Relation where the relation can read the data's value.
- Writer Connection: A Connection between a Data and a Relation where the relation can write a new value in the data.

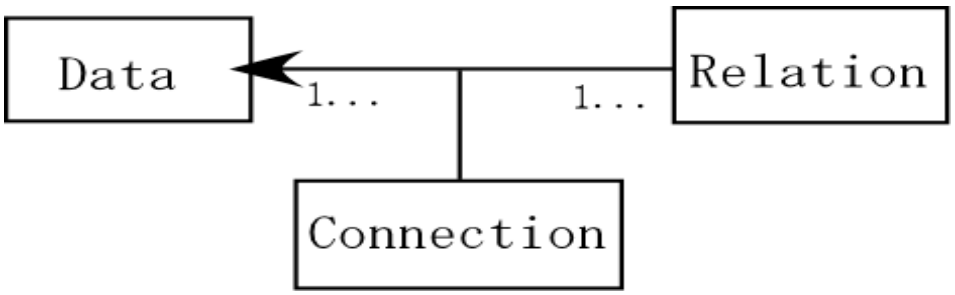


Figure 3 - An entity/ Association diagram representing the relations between Nodes and Connections

Nodes can be linked with as many Connections as needed but a Connection, whatever its type (Reader or Writer) always connects a Data Node to a Relation Node.

The example below will show you a basic representation of an FDNetwork to help you to visualize how its components are organized in order to create Robot's intelligence.

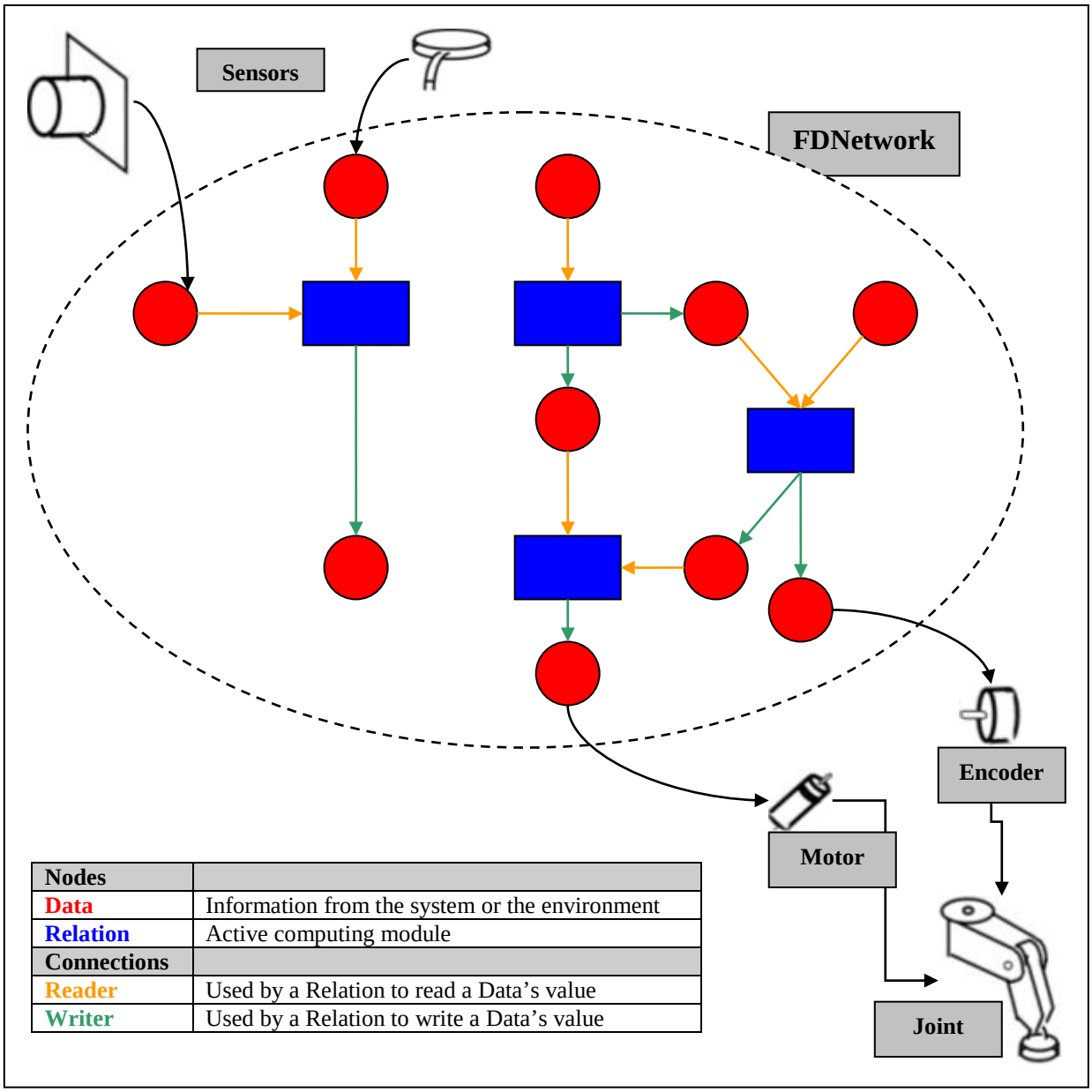


Figure 4 - Basic representation of an FDNetwork

The example below will show you the dynamism that exists in FDNetworks. It represents a sub-network that can generate the motion of a robot using the cooperation of many movement formation agents. The movements dynamically created are computed basing on the information received from different kinds of sensors, whose aim is to provide real-time information about robot's condition.

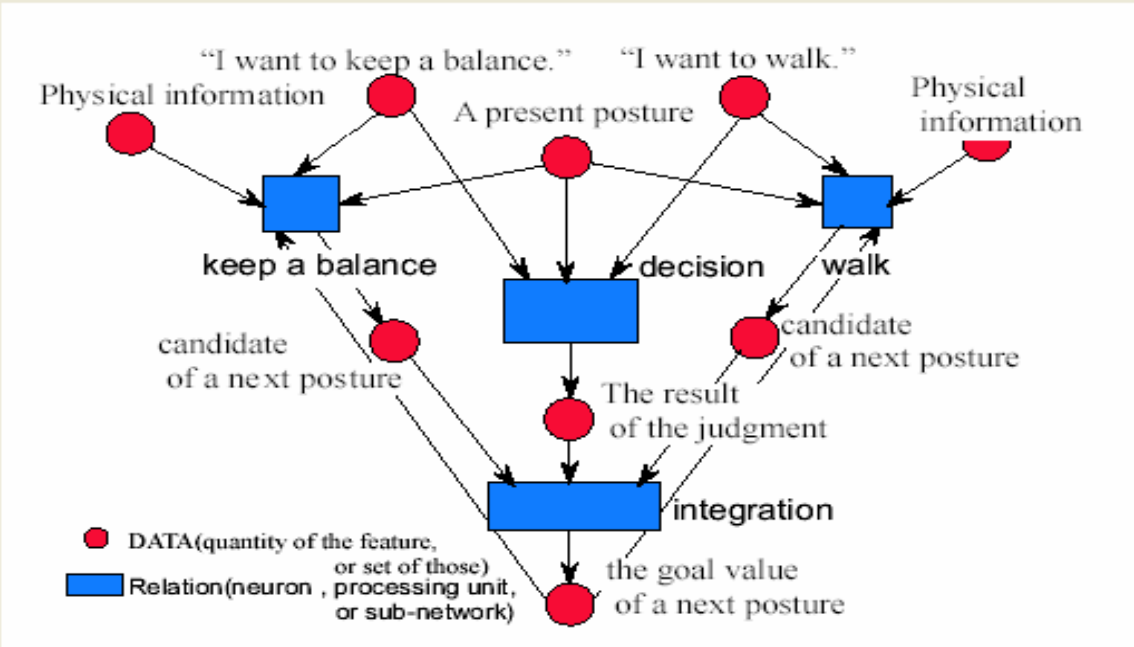


Figure 5 - A sub-network that can generate the motion of the robot

The Datas in entry of this sub-network ("Physical information", "A present posture") give information about robot's current physical state and posture. Aside from theses Datas, coming from real-time sensors, the sub-network also needs to know robot's intentions ("I want to keep a balance", "I want to walk") in order to decide robot's next posture.

Theses Datas are read by different Relations ("keep a balance", "walk") who independently calculate a candidate value for the next posture. Theses candidates will, along with a Data representing the result of robot's judgment about moving or not, will be integrated to calculate the next posture the robot has to take.

Note that all is not necessary black or white. The decision can be, for example, that the robot has to move, but just a little. In this case, the integration will give a higher importance to robot's balance but will not just stand still for all that.

In fact, as all Relations are just programming, anything wanted can be computed. The main problem faced by researchers is not to compute Datas the way they like but to possess the right information at the right time and to know exactly what to do with it in order to create a usable output.

In the example above, it is important to note that "The goal value of a next posture" is, in fact, the same data as "a present posture". This Data just receive a new value from one of its "child" Relation.

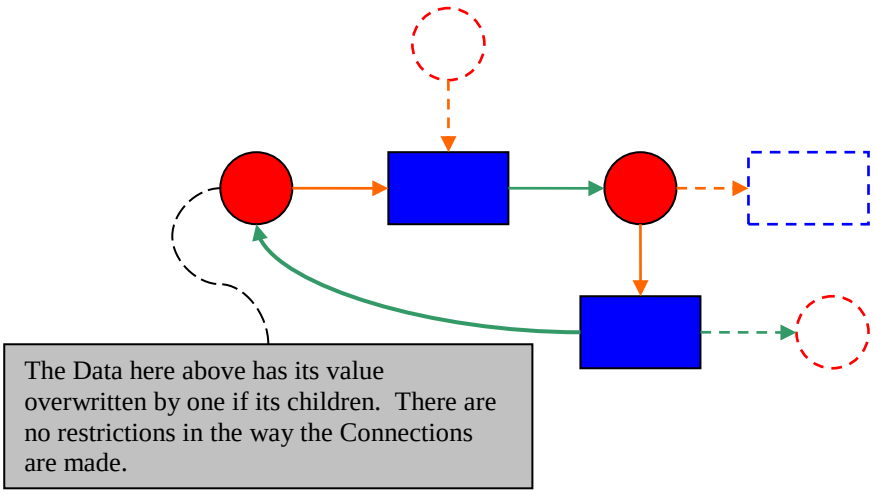


Figure 6 - FDNet's "Flat" characteristic

This fact represents the "**Flat"** characteristic in the definition. The Networks are said to be "Flat" because, through the use of Connections, any Relation can be connected to any Data, whatever their meaning. Inside an FDNetwork, there is no hierarchical structure or different component levels. All Nodes and Connections have the same status in the Network and are processed in the same way.

Self-organization comes from this fact. New features can be dynamically constructed using information coming from any Node in the Network, whatever the type of information they bear.

The term "**Distributed**" is used to point the fact that, in FDNet, a group of robots and machines can work together from the intelligence of a single Network. It means that each robot doesn't especially have its own FDNetwork but can share it with other robots, all contributing to Network's global intelligence.

For example, in the Figure 6 - FDNet's "Flat" characteristic, the flying robot can provide information to the network that will be very useful for the crawler robots to narrow their searches.

2. FDNet is based on the Human Imitation Model

Before the specification of FDNet's architecture, the researchers have adopted a recognition model. In FDNet project, the outline of human imitation is assumed for the rescue robots to solve most of the problems. They have done many researches in which they transposed some human features to a robot and studied the application to the robot. Next is an overview of this research.

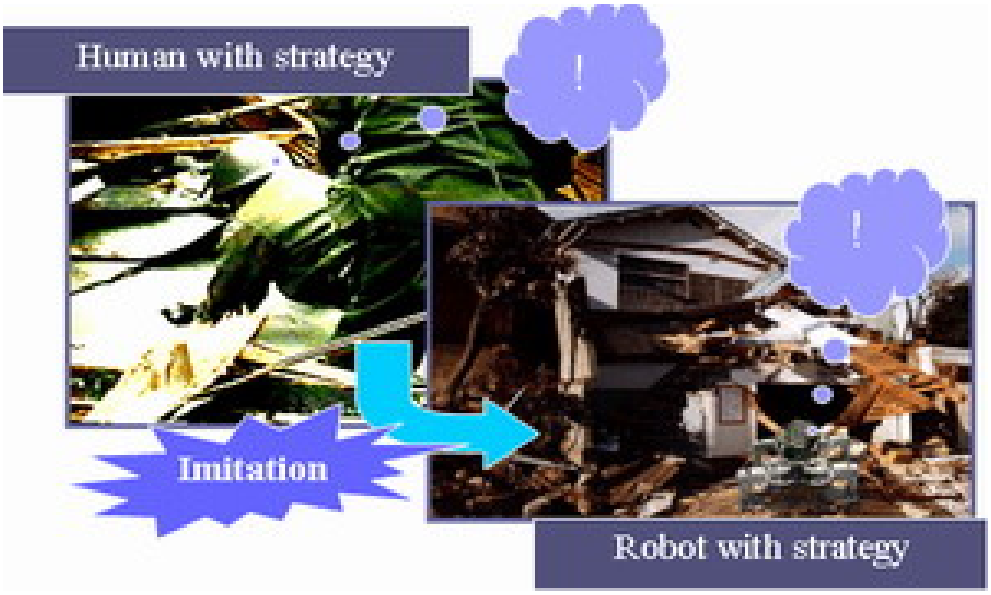


Figure 7 - Human imitation model

Human Imitation Model is used in FDNet for more than one purpose. First of all, Network's intelligence is based on the neural model, meaning that human intelligence is, to a certain extend, imitated by FDNet.

After researches about what rescuers do when working in devastated environments, I.R.S.I. Researchers found that humans do two particular movements when moving around to find victims: first, they tap the ground where they are planning to move in order to see if it is stable and after, they slowly move their weight forward and analyze their own position in order to be sure that the new position they are in is stable too.

Following the "Human Imitation Model", I.R.S.I researchers tried to transpose this information about movement inside FDNet model. Conclusions made clear that it is not only important for a robot to move around to imitate human rescuers actions but also, it is necessary to take care of the way it moves and to analyze the "feeling" he has about its own position in order to take the best movement's choices.

To make the best work possible here was quite important because moving on devastated environments is one of the main tasks of FDNet robots. If this task

cannot be performed correctly, none of the following ones will be possible to achieve.

In order to achieve this requirement, FDNet model was expanded to include an “Active movement sense”. This means that, gathering information coming from sensors (tapping the ground in order to see if it is stable), motion, intentions and environment, FDNet robots develop a perception-like ability. Their movements thus become more precise.

The information acquisition and recognition was realized by creating an “FDNet neuron formation” efficient for the specified purpose. This formation can be updated each time a common recognition model between robots and humans can be found.

Note that what are transposed are the high-level tasks humans execute. By doing this instead of transposing low-level ones, researchers can enhance their robots with abilities specifically thought for them. For example, four-legged robots (such as the one you see in the picture below) will never move the same way as human being do even if the intentions about moving are the same.

This is where self-organization can show its true power. It can become the base for great enhancements because it can produce features that would be very difficult to find while examining Human beings. Indeed, if it is quite easy to create a general scheme of the way a human being moves, it is a lot more complicated to find all the specificities of the same movement.

Basing on the transposition of high-level Human tasks, self-organization will be able to reconstruct a “Human-like model” by finding an organization which allows producing the same results as a human being.

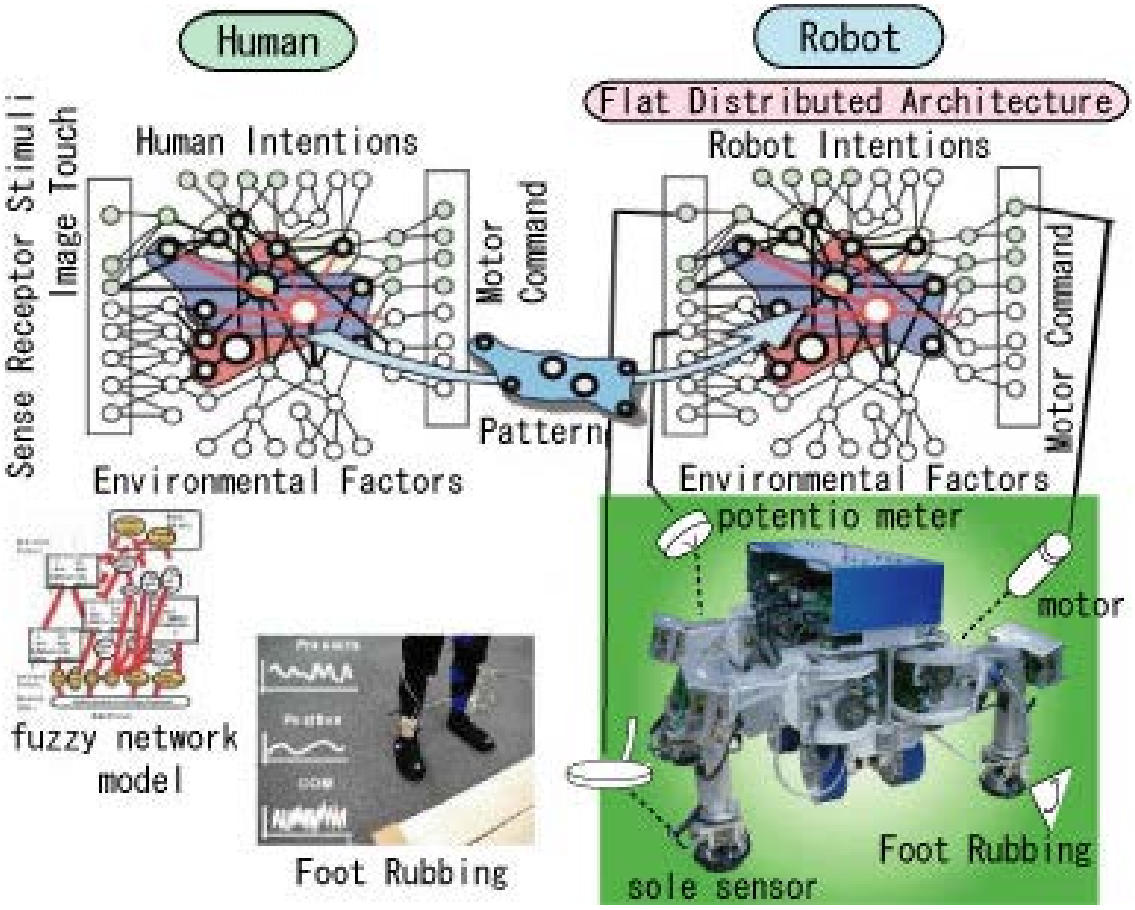


Figure 8: Human like

Figure 9 - The RoQ imitating human action [3]

3. FDNet's aim is to help rescuers to find victims in case of disasters

FDNet is, as said before, a common architecture especially created for rescue robots. Indeed, previous studies on robot architectures were for entertainment (Open-R) and industrial robot (ORiN and ORCA) but none of them could be easily transposed to be used with rescue robots, theses ones having particular needs.

As rescue robots are the base of the architecture, FDNet has been specifically created to ensure it can respond to theses robots needs and thus is able to solve the important problems presented first[3].

4. Current FDNet implementation

First of all, the creation of a complex rescue robot has been started some years ago and is still under development. The rescue robot that can be seen here below, named RoQ (Robotic Platform for Rescue), serves as the base platform used to construct FDNet and test its abilities.

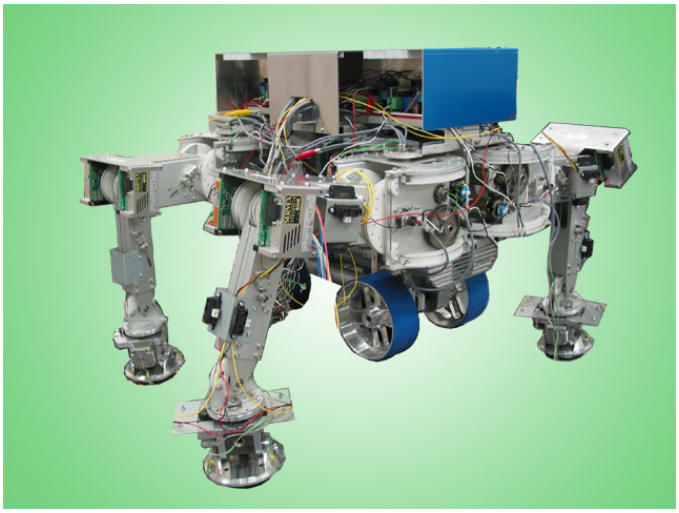


Figure 10 - The robot RoQ

FDNet implementation started at the same time. It is still a work in progress and much time will be necessary to have it working perfectly, though a beta version is already showing some of its potential. But as far, the Robot is still not working.

[3] Report to FDNet's concepts in this chapter.

5. Development choices

a) RoQ's base Hardware

- Quadruped robot TITAN-VIII
- PC (Pentium-III 800MHz, 512MB RAM)
- Device Network
  - Angle of inclination meter, Infrared rays sensor
  - Ultrasonic sensor, CCD camera
- Tactile sensor at the sole
- Wheel movement mechanism
- Ankle mechanism

b) FDNet environment

- The Linux operating system (kernel Linux 2.4.4)
- Real time extension RTLinux 3.1
- Java language(Java2 SE 1.4.1 01)
- postgreSQL DataBase (V. 7.1.3)
- CORBA Middleware (OpenORB 1.2.0)

c) FDNet A.P.I

Without entering too deeply in the details, we can say that FDNet's API is defined within the following layers:

- The Network layer (CORBA): It is the place where Connections and information transmission are implemented.
- The Programming language layer (Java, C++): It is where the real functions and behaviors of the Relation and Data objects are implemented.

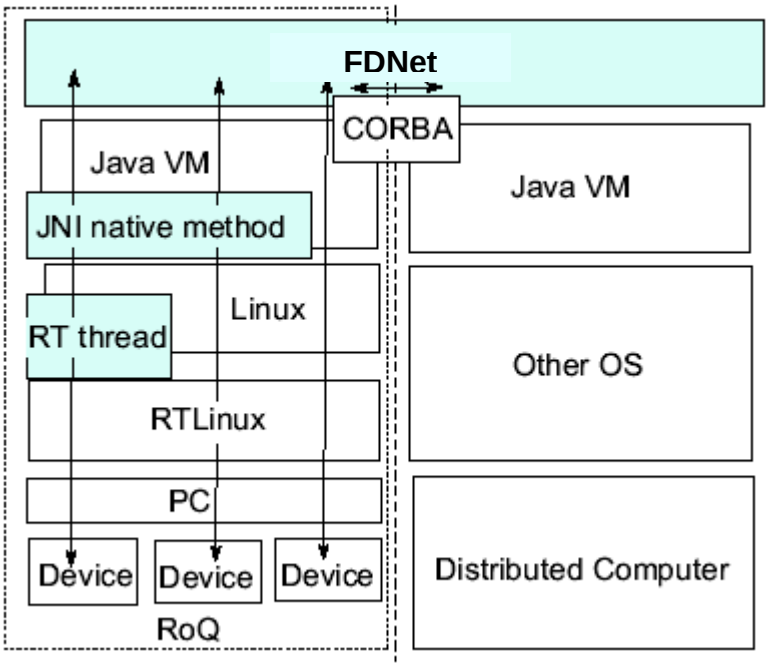


Figure 11 - Implementation of FDNet in various layers

While Java/CORBA was selected because of Java's portability feature the implementation on RTLinux is used to control the parts that are time critical (i.e. a defined response time is expected). For example the control of each RoQ Robot's joint is mounted as a real-time task of the RTLinux and can be available directly for the other components.

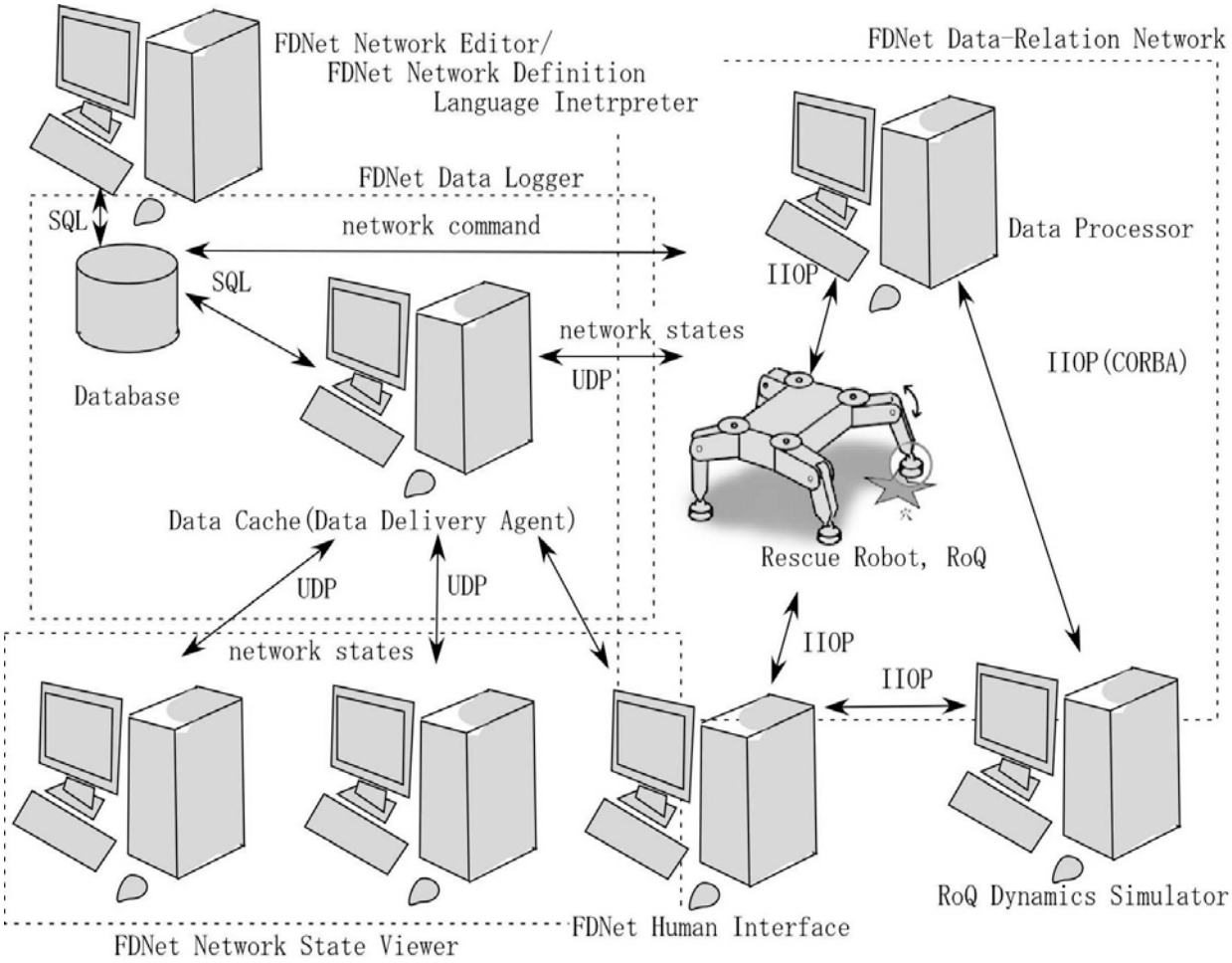


Figure 12 - General view of FDNet that show the relations between the different applications of FDNet

C. State of the Art

FDNet's architecture is based on 3 main architectures: ORiN, Open-R and Orca. FDNet researchers have deeply studied and analyzed theses architectures to find their advantages and weaknesses in order to improve FDNet's conception. We will also describe CLARAty because of his reliable reputation.

The main ideas of theses three architectures will now be presented to give you a glimpse of FDNet's origins and working.

1. ORiN: A common object model for robotic systems[4]

As a three-year project of NEDO (New Energy and Industrial Technology Development Organization), JARA (Japan Robot Association) started "the Development of a standard interface that provides a unified access mean" from 1999. The outcome of this project was ORiN (Open Robot interface for the Network). In other words, ORiN is a system for standardizing communications interface between personal computers and robot controllers.

In general, robots' accessing methods differ from manufacturer to manufacturer. By standardizing this access, ORiN transfers data[5] stored in the controllers of various industrial robots onto personal computer. Once transferred, this data can be easily accessed via networks and shared among application monitoring robot operations, equipment diagnoses and even for production control.

ORiN provides the following advantages:

- Uniform data exchange is possible between robots created by different manufacturers.
- ORiN being an Open specification, conform application can be developed by third parties.
- Ease of configuring multi-vendor systems.
- Worldwide standardization through proposal to ISO.

ORiN expected to bring about the following economic effects:

- Increased competitiveness in manufacturing.
- Expansion of the robot market.
- Entry of the software industry into the robot market.
- Creation of a robot engineering industry.

To achieve the above-mentioned objectives, it was decided to configure ORiN with provider, kernel and application logic layers.

The provider layer compensates for the differences in expression and/or protocol of robot controller data among various manufacturers and transfers data to

[4] See [JARA 1999], [Inukai 2003]

[5] For example: Robot's position information, number of parts to be assembled, number of defective parts, etc…

the kernel layer, which is configured based on RAO (Robot Access Object) and RDF (Robot Definition Format).

RAO applies DCOM distributed object model technology to provide network transparency and uniform robot access, while RDF uses XML to provide files for defining structural models of robots with expandability. This enables ORiN to accept individual robot differences thus allowing it to be continuously used in the future.

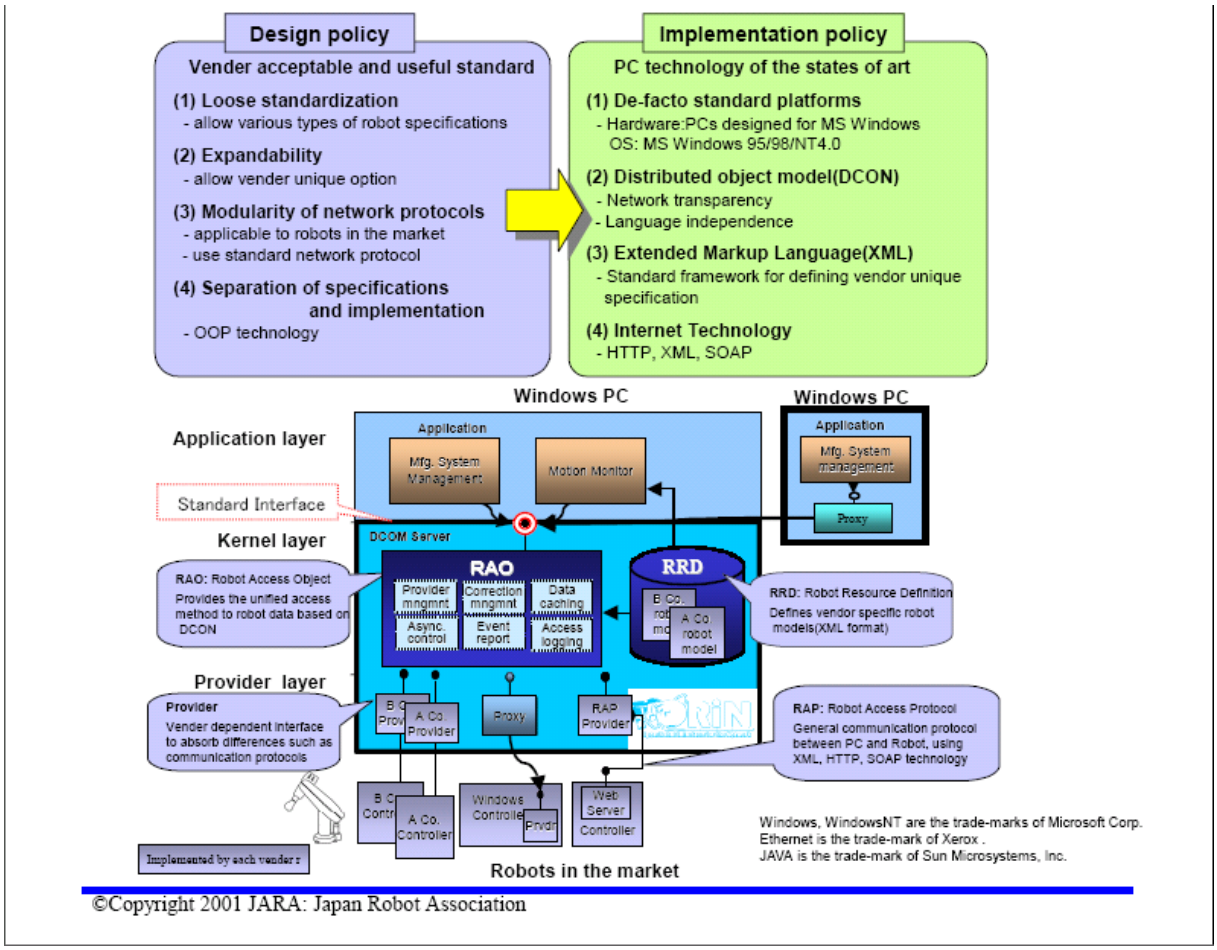


Figure 13 - Scheme representing ORiN's implementation.

2. Open-R: An Open Architecture for Robot Entertainment[6]

Sony Corporation has proposed an open architecture for autonomous robot systems, which aimed particularly, but not exclusively, entertainment applications. In order to achieve system extension and reconfiguration capabilities for mechanical, electrical, and software systems, they have proposed an architecture with the following features:

1. A common interface for various components such as sensors and actuators;
2. A mechanism for obtaining information on functions of components and their configurations;
3. A layered architecture for hardware adaptation, system services, and application providing efficient development of hardware and software components. A software platform provides an environment for agent design so that designers can customize their recognition and control algorithms. This is based on Apertos, a fully object-oriented real-time distributed operating system which allows each physical and software component to be defined uniformly as an object.

The outcome of this project is Open-R and its goal is to establish a draft standard for mobile robots and their software systems. This standard would allow different companies and researchers interested in entertaining robots to build their own products and prototype systems using readily available components which meet Sony's specifications.

Their open architecture and standard target entertainment applications for three reasons:

1. **Complete Agent:** A robot for entertainment requires a complete autonomous physical agent. Instead of research and development activities focusing on specific perceptual functional components such as speech and visual cognitive subsystems, a complete agent promotes and accelerates research activities involving combination of subsystems and whole robot systems.
2. **Technology Level:** Robots for entertainment applications do not require such high performances in speech recognition and visual information processing that are required in mission-critical industrial applications. While there exist special and difficult requirements in entertainment applications themselves, limited capabilities or performances can cause a certain kind of excitement to users in most game playing situations such as RoboCup (soccer games by robot agents). This implies many existing AI technologies can be implemented for these kinds of applications.
3. **Emerging Industry:** Sony's researchers believe that they will be able to create a completely new market in the near future by introducing this kind of robot product sharply focused on entertainment applications. After the Gold Rush of Internet and cyberspace, people will eagerly seek real objects to play with and touch. Robot Entertainment provides tangible physical agents and an undoubted sense of reality.

[6] See [Fujia & Kageyama 1997]

By establishing a standard for entertainment robot software as well as robot parts, manufacturers can produce and sell their own commodities using the standard. AI researchers often spend large amounts of time customizing hardware. Readily available components allow researchers to construct customized robots for their research platform minimizing time consuming hardware and software hacking.
Below are Open-R system architecture's main features:

- **Open Architecture:** Open-R defines a set of standard interfaces for physical and software components and a programming framework, so that anyone can design extensions to the basic robot system within this standard.

- **Configurable Physical Components:** Open-R defines a common interface for all robot components for flexible and extensible robot configuration. This includes a mechanism for obtaining information on component function and configuration for interactive applications. Along with object-oriented software architecture, the Open-R provides Plug-and-Play capabilities for physical robots.

- **Object-Oriented Robot OS:** Open-R employs Apertos, a fully object-oriented distributed real-time operating system. This enables to define all physical and software components uniformly as distributed "objects".

In Robot Entertainment, there will be various applications, such as a pet-type robot, a game-type robot, or a tele-presence robot, which may be fully autonomous, or remote controlled semi-autonomous.

For these applications, the following are considered to be common requirements:

- stand-alone application
- extensibility
- friendly application development tools.

Figure 14 - AIBO, the SONY robot under Open-R

3. Orca: Open Robot Controller Architecture

To make robot technology becoming widely used, and to make various robots appear in the market, robotic parts - including mechanism, hardware, and software - should be produced as components with open interface. A lot of activities have been done on research and development for robot technologies in the world. However, the robot technologies so far cannot be reused because of incompatibility of the robotic parts. With the open interface, it becomes possible to use the robotic parts to build a wide variety of robot systems. Some people are confident that robotic technologies and know-how's can be accumulated for reuse with these reusable robotic parts.

To reach this hope, the Toshiba Corporation researches leaded to the creation of Open Robot Controller Architecture (ORCA[7]) which allows making the reusable robotic parts (software / hardware) to enable easy built-up of robot controllers. ORCA also allows manufacturers to quickly and easily integrate robotic parts developed by third parties into their systems, achieving efficient development of advanced robots in a relatively short period. Speech processing systems, image processing systems, robot control systems, and so on, are easily mixed up to build a robot system.

Toshiba's researchers have proposed Robot Technology (RT) reference model as an RT software layer structure. The structure consists of five layers:

- The physical layer;
- The I/O link layer;
- The actuator control layer;
- The motion control layer;
- The task layer.

ORCA has been defined according to the RT reference model. With the RT reference model, a developer can concentrate on a layer for which he develops software, because the software can utilize software for the lower layers with the open interface.

ORCA utilizes distributed object technology to abstract the communication between robots and between components. The distributed object technology enables developers to program a whole robot software system in object oriented manner. In ORCA, we can treat all robots as objects, and all the objects are defined with open interface. With the use of the distributed object technology HORB, the robot objects can be distributed anywhere in networks and they can be used directly from any node in the networks.

ORCA also consists of various interfaces and classes containing robot control software and acting as the ORCA's API specification. In order to use ORCA, developers have to implement the APIs defined in the interfaces. This ensures that any ORCA user will be able to use ORCA-based controller created by any other developer in the world.

[7] See [Ozaki 2003] and [Toshiba Corporation 2003]

4. CLARAty: Coupled Layer Architecture for Robotic Autonomy

CLARAty [CLARAty] is a collaborative effort of several institutions, especially because the project also belongs to the Mars Technology Program and the Intelligent Systems program. The main CLARAty's aim is to provide reusable and flexible software for the current and futur rovers. To reach this objective more easily, the software is distributed through an open source license. Currently, CLARAty is running on the Rocky 8, FIDO, K9, Rocky 7 and the ATRV rovers.

Its architecture is divided in two layers: the Functional layer and the Decision layer (Figure 15 - CLARAty's layer).

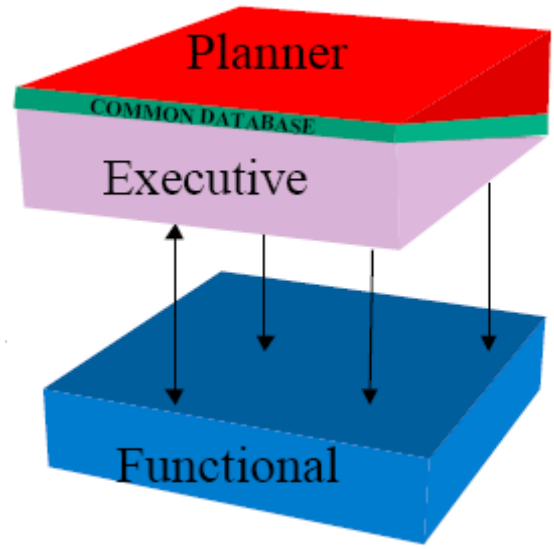


Figure 15 - CLARAty's layer

The Functional layer provides several generic framewroks centred on many robotic-related disciplines. It is composed of four main features. The first is a decomposed level with various degrees of abstractions. For example CLARAty provides a locomotor's interface that suits all the wheeled or legged rovers. The aim of the second part is to separate algorithmic capabilities from system capabilities. That way the system limitations and the algorithmic limitations are separated in order to avoid propagation of assumptions that are unique to a particular platform. The third feature has some correspondences to the second part. In fact its objective is to separate two points of view. Here the separation affects the behavioral definitions and interactions of the system from the implementation. This separation allows the dynamic binding of adaptations at runtime and also makes both the functional and implementation trees extensible. Fourth, a flexible runtime model is provided. This runtime belongs to the abstraction model. This model is associated with the generic functionality and with the adaptation. So it allows the runtime model to be dependent on the hardware's capabilities and at the same time it provides the possibility to change from a system to another.

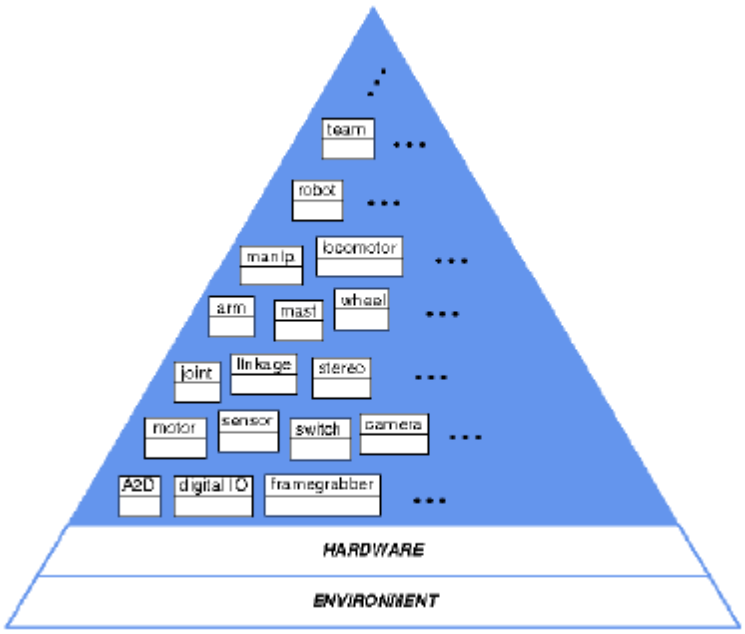


Figure 16 - CLARAty's Functional Layer

The Decision layer is the top brain of the software. It manages the functional layer to achieve the goals through a general planner, schedulers and resources managements. This layer interacts with the Functional Layer using a client-server model. The server (the Decision Layer) asks the client about the system resources and sends commands. The planner's implementation is the CASPER planning and scheduling system and the executive's implementation is the TDL executive system.

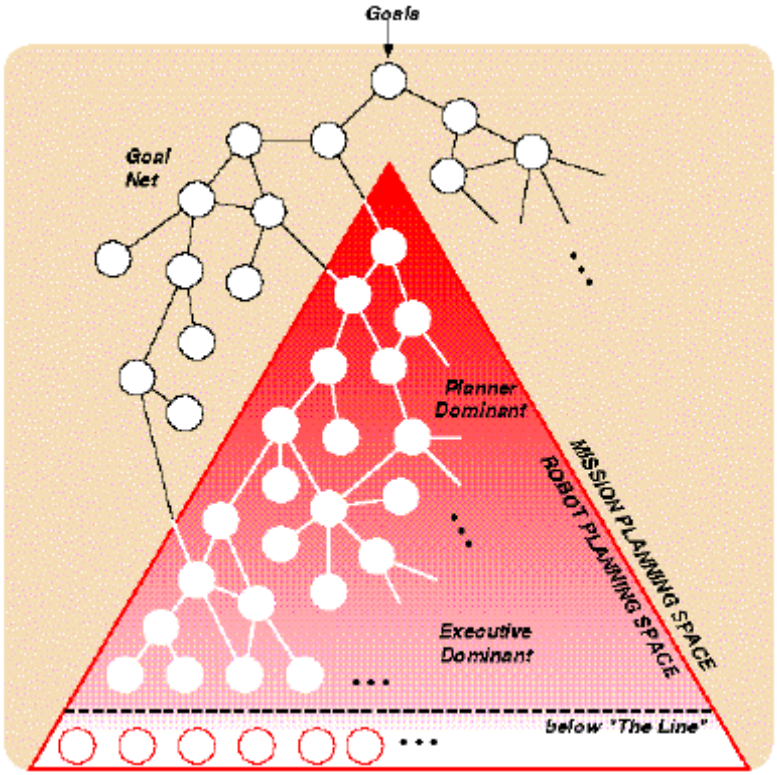


Figure 17 - CLARAty's Decision Layer

This design provides to CLARAty the capacity to give to the rover the needed autonomy on the field and to fit to different kind of hardware without any additional program. But only one rover could be managed at a time.

D. FDNet Positioning

The following FDNet's definition gives you an idea of FDNet's aim, its capabilities and its potential power. Though it is still far from being a completely usable architecture, its implementation is advancing well and tests that have already been made, mostly about human imitation model, are satisfactory enough to give a strong will to continue its development.

ORiN standardized the specifications of interface and data, as well as standardizing negotiation for robot controllers to communicate with various applications. This research allows heterogeneous robots to communicate with each other, and also separates applications from robot controllers.

OPEN-R is a re-configurable robot platform for the robot's entertainment's system. This platform analyzes a robot configuration in syntax and gives semantics to robot's components. Based on OPEN-R, a program is portable to robot hardware without changing codes.

FDNet is then very promising. However, the flat's "F" is going to lead to some kind of problems, mostly if different people are, meanwhile, using the same network. Indeed, via "flat" anyone (any Relation) is able to write in a Node (i.e. Data). There is no restriction's mechanism of access right. It could be useful, for instance, to lock some acknowledged patterns of nodes in order to prevent a future modification to fetter the network to work properly. If we want to ground on the FDNet pattern's concept, we should integrate it in the DTD. The graphical user interface could, for instance, show the nodes of a same pattern in a same colour. At last, the idea of allowing a user to change some network's worth by himself seems pretty complicated: why then could not we run it with a remote control?

In the near future, the test robot – RoQ – will be improved and more and more capable prototypes will be created. At the same time, FDNet will have its bugs corrected and the tests between the network's architecture and the robot will lead to providing answers to some of the questions at the base of FDNet's development.

Later on, FDNet's distributed component implementation will allow seeing how a pool of robots using the same intelligence can perform and will give life to a lot of future researches. As now, IRSI is trying to run one unique robot under the control of an FNDet. But it is completely different to manage a pool of robots by only one network. The writing of a network seems to be congealed, static. As far, FDNet does not have any mechanism of network's dynamics: it is still really hard to adapt the network to any changes. How could we add/remove a robot from the network without having to add/delete every single connection manually? We can conceive that a network working with only one single flying robot acting as an eye would behave the same way without this one. We would have to ask ourselves if whenever we build a network with only one robot, should we not plan the eventual presence of others? In most networks you can find an intermediary entity taking care of this, not in FDNet. Often, robots have at their disposal private Message Channels. So, the idea would be great to managing one unique robot, even if we can still find it

hard, but once you want several designers to manage a pool, the work would be totally different.

III. The Human Interface

A. Aims

One of the main problems faced in Robotic Rescue Systems field is that the people creating the robots (electronic scientists) have to learn skills which are separated from their field of work (computer skills). They have to do it anyways because, in general, teams are too tiny and only composed of electronic scientists. Electronic engineers should only have to bother about creating robot abilities and design the way they work using applications handling all computer related specificities for them. This would allow them to work faster (lifecycle would be highly reduced because lots of problems would be solved automatically by the application) and better (specialists would only be asked to use their specific capabilities).

As said in the introduction, the FDNet technology is powerful but quite complicated to use. In the viewpoint of FDNet's creators, the technology is to be used by a lot of people, ranging from computer scientists (of course) to electronic scientists, which, even if they know some things about computers, are perhaps not at ease with every piece of knowledge needed to create an FDNetwork from scratch. As now they are using huge XML files (more than 20 pages) interpreted and executed by complex command lines. It means that if they wanted to test a special part of their robot (ROQ), they are forced to write down a huge quantity of code just to make simple tests. This had some serious implications:

- Users could not easily test robot's additions or modifications.
- Due to the amount of work required to write an FDNetwork, even a little one, more complicated tests were impossible to handle without an application helping the tester.
- Writing a complete FDNetwork including robot's full intelligence was impossible, or at least, working on an efficient intelligence was unfeasible.

H.I's main goal is to hide all computer related aspects of writing FDNetworks to allow Electronic engineers to focus on their own work. Giving a simple way for FDNet users to edit the FDNetworks allows them to manage the robot(s) before and while they are functioning.
Editing FDNetworks easily allows electronic engineers to:

- Reduce the Lifecycle of robot prototypes by allowing them to test directly the physical modifications they are making to the robots.
- Enhance robot capabilities by giving the power to create more advanced networks in a highly reduced time.
- Create more robust networks by hiding them their complexity and by controlling the actions available for every part of the network they are working on.

The H.I. is then a very important addition to FDNet and its use will allow FDNet to fulfill its greatest contract: Providing rescue robots engineers with a completely distributed network without forcing them to acquire new skills, unrelated to their own work.

B. Human Interface appearance

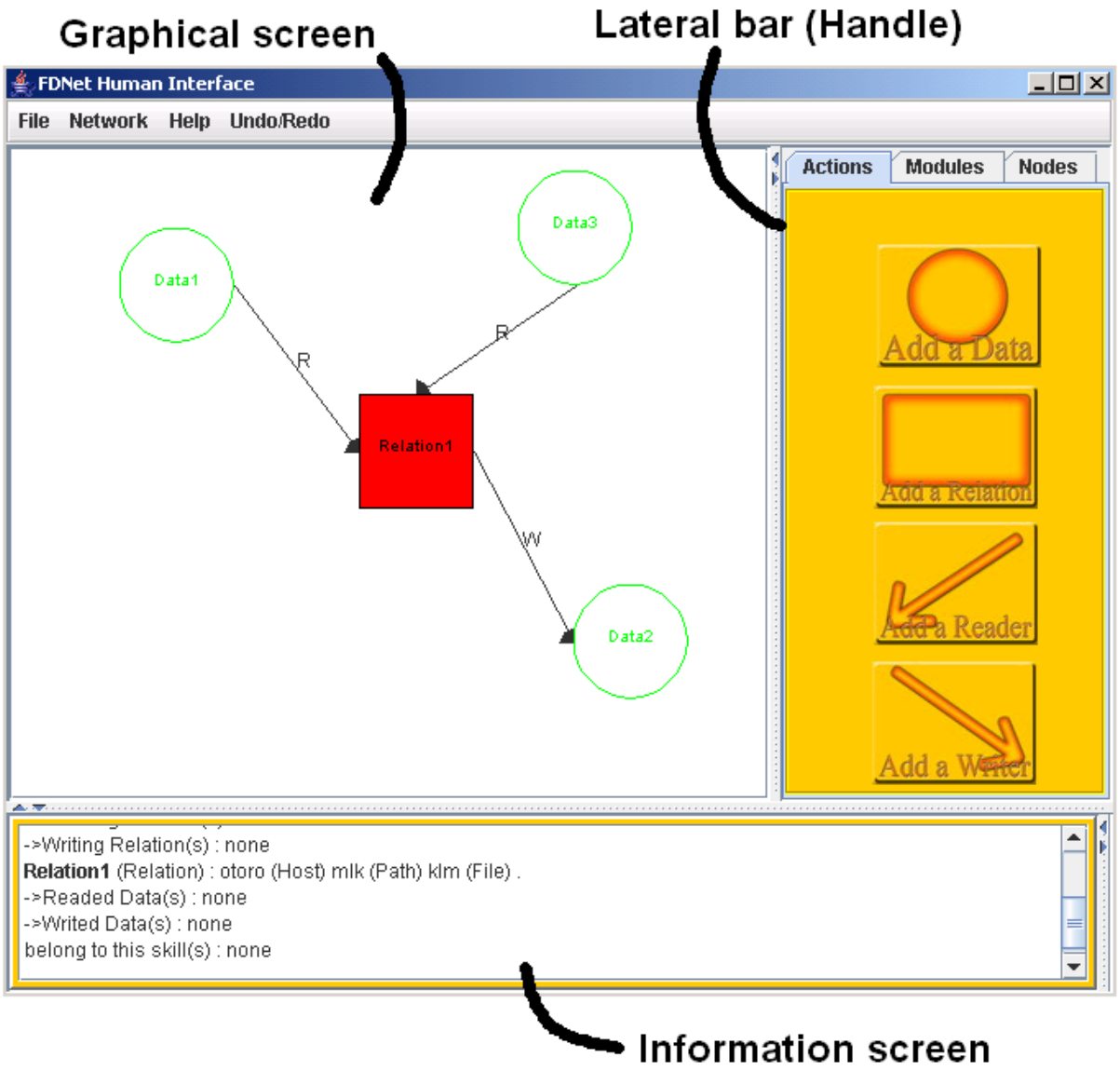


Figure 18 - The Human Interface appearance

The user drags and drops components from the Lateral bar to the Graphical screen to add them. He can then move the elements and arrange them as he wants. The Handle provides ways to manage a module, to add/update/delete a node etc... The Information screen logs all the action the user does and gives sometimes more information on the action done.

C. Functions

The aim of the Human Interface is to give a simple way for FDNet users to manage the robot(s) through an easy edition of an FDNetwork before and while the robot(s) are functioning. Before: by providing a case-tool for creating and editing networks. While: by providing way to monitor the evolution of the network and by permitting to introduce new values dynamically.

The FDNet Human Interface is the visual shell of FDNet. It enables to design network structure graphically and send the information to the FDNet. It enables to convert the network's structure which FDNet has in itself and to observe values of entities in FDNet.

D. Use

The Human Interface (HI) allows to manage an FDNetwork. As said before, the network may be composed by nodes (data or relation) and connections (reader or writer). To improve FDNet scalability and utility, a new layer was introduced [2]: the module extension. A "**Module**" is a set of nodes and connections that can be loaded or not at the start of the network. This new extension permits to work on a specific part and to divide the network into smaller, clearer pieces. Note that thanks to this new feature, two entities with the same identifier are able to exist in two different modules.

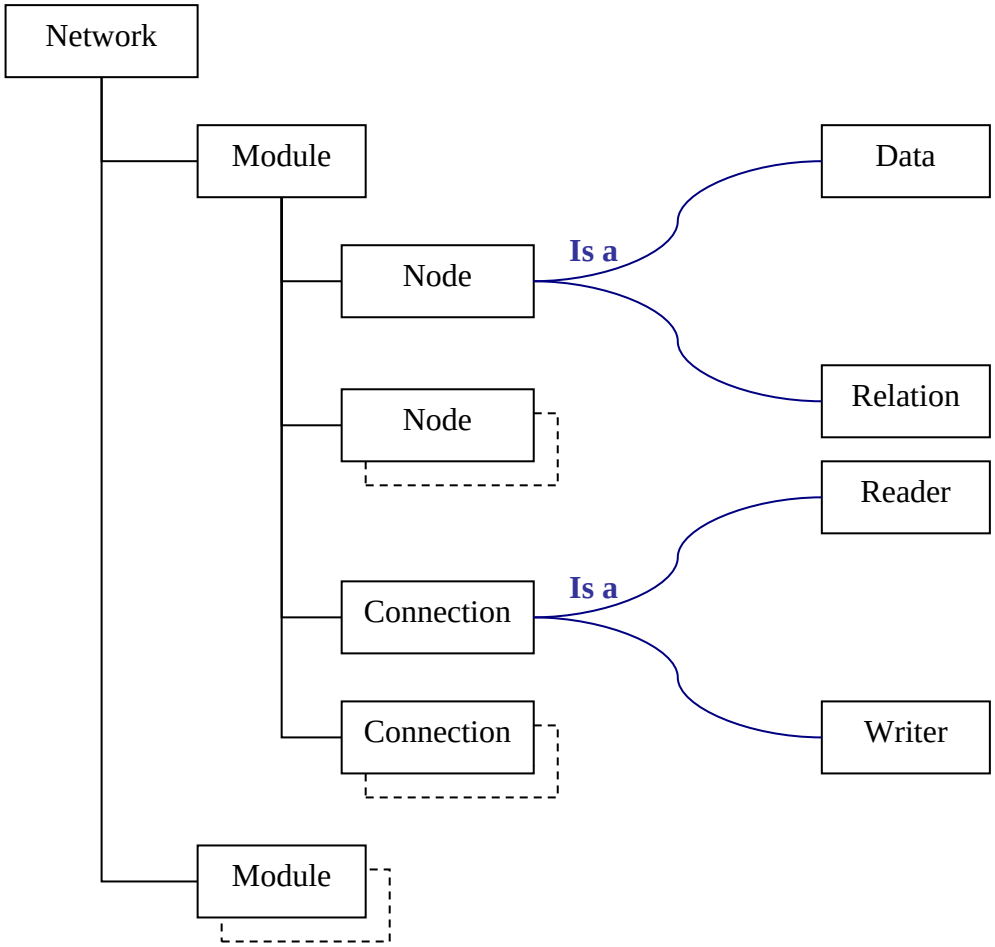


Figure 19 - The NetworkInfo system architecture

E. Human Interface architecture

The Human Interface consists of 2 different main parts. Theses are:

a. The logical base: this part consists of the Network Repository and its access system. This is the base of the architecture. Its aim is to handle all the low level tasks that have to be performed in order to be able to interact with FDNetworks.

b. The end-user part: While the first part only focuses on handling low level access to FDNetworks, this part will handle all the interactions with the end-user. In fact, this part consists of what is generally called the front-end. It will be divided in three specific points:

- The static capabilities;
- The dynamic capabilities;
- The display of all this information in an easy, yet powerful, way to understand.

1. Specification of the logical base

The first thing that the Human Interface must be capable to do is to save user-created FDNetworks to allow them to reuse the networks at a later time. But to reuse a network does not only mean that the Human Interface has to be capable to load/save it. More than just this, that information loaded has to be easily usable at work-time; it means that, when the Human Interface displays graphics, all the content behind them has to be fetched in a single logic entity.

a) FDNetwork specification

The FDNetworks are stored in an XML file in the specific XML format described in the annexes. The system we used to share all FDNetwork information in work-time (i.e. inside the Human Interface) is called the “NetworkInfo”.

b) NetworkInfo system functions

As explained here above, the “NetworkInfo” system has been created to share all the logical information about an FDNetwork inside the Human Interface. It can be seen as an intermediate layer between the FDNetwork repository (the XML structure) and the Human Interface. Its aim is to handle all operations on the repositories, thus saving and loading FDNetwork definitions.

The Network level is the most general one. Aside from containing all information about the FDNetwork read from the repository and services applying to the network itself, it also contains all the services applying to more than one module at a time.

Following is a set of services that can be found at this level:

- Add/delete a module to/from the network;
- List all the modules found in the network;
- Change the name of a module;
- Replace one node by another one (handling integrity constraints related to the node replaced)
- Find all connections linking a specific node (search is made inside all modules the network is made of)

The Module level allows dividing an FDNetwork into more tiny parts, each of which can be treated separately.

At this level, you will find all information related to one specific Module:

- Module's name;
- Its load modifier (Does this Module have to be loaded in memory when user starts the network or not?)
- Number of data, Relation, Reader and Writer contained in the Module

You will also find here all services related to gathering of Nodes and Connections, such as:

- Retrieve information about a Node/Connection existing inside the Module.
- Retrieve all Connections related to a specific Node (existing inside the Module).
- Delete a Node/Connection existing inside the Module and, in the case of a Node, delete all the Connections related to it too.

Nodes and Connections, apart from being the most specific levels in an FDNetwork, are also the only ones containing "real" information about the network itself.

Network and Module levels are only structural levels: they allow putting some order in a network but they do not contain computational informations. The levels that are really modeling an FDNetwork are the **Node and Connection levels:**

- Data Nodes contains the code allowing to fetch (from sensors) or create (from information inside the Network while it is working) values and to provide them to the Relation Nodes.
- Relation Nodes contains code related to Robot's intelligence. They read values found in the Data Nodes and compute them in a specific way to create new Datas as output.
- Reader and Writer Connections are the links between the Data and Relation Nodes. As such, they contain information about how the links have to be done, the way the links work and so on.

Services related to gathering information about the Nodes and the Connections can be found in these levels.

It is also interesting to note that Nodes do not know links they have with other Nodes. All information about linking is to be found in the Connections. Nodes can only provide information about themselves but Connections provide information

about the two Nodes they are connecting (a Data and a Relation). This is an important fact since it has serious implications on the whole architecture.

2. The end-user part

Structured in three levels as the NetworkInfo, the graphical part does not store the semantic information of the nodes but deals only with the static capabilities. It is in charge of all the states of the nodes like the XY position on the Graphical screen and its color for example. The XYLayout manager from Borland/Inprise JBuilder© handles the graphical components. The dynamic capabilities are managed by another package (see [Lambot]).

3. UML diagrams

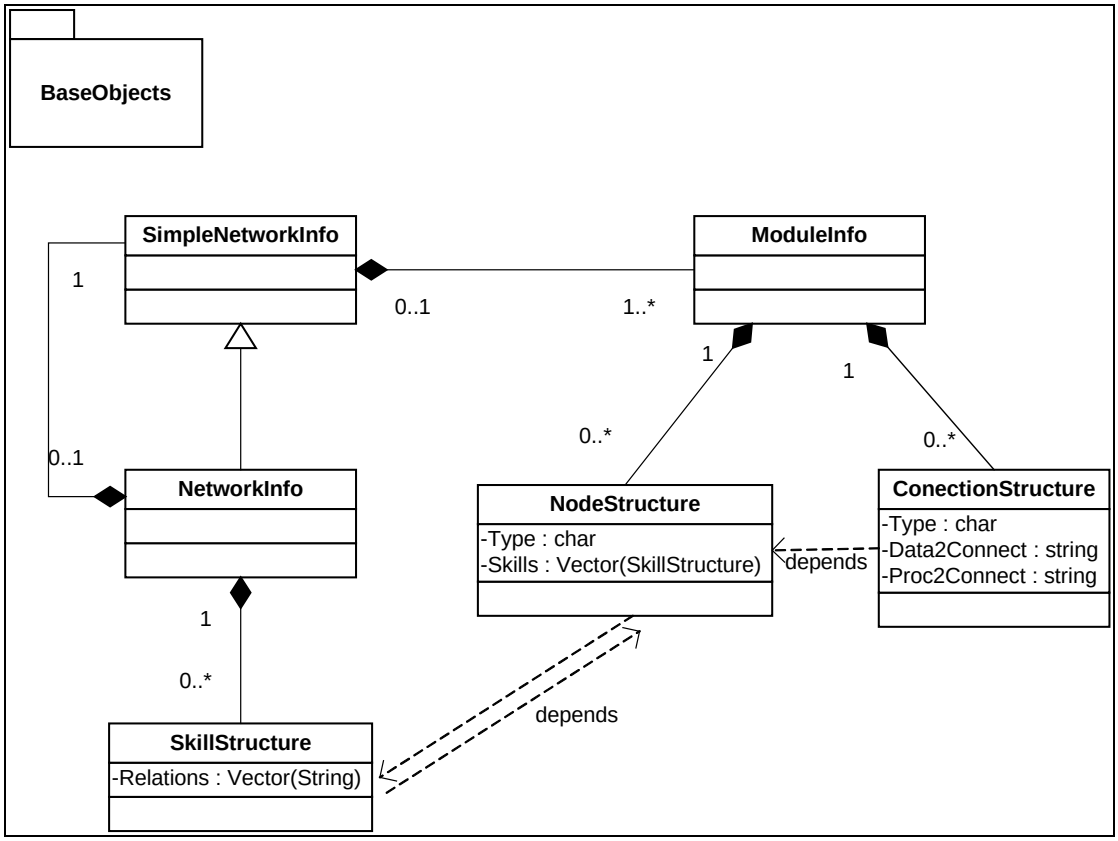


Figure 20 - UML Class diagram of the logical part

The architecture separates the logical from the graphical part of the network. The logical part is saved in a .fdg file (built as an XML file) and the graphical one in a .fdn file (with the Serializable class of Java). The graphical part deals with the displaying of the network in the Graphical Screen. The main class is mainfiles.mainProgram. This one creates the logical part of the network (baseObjects.NetworkInfo) and the main window (Forms.MainForm). This window will create the graphical part located in the GraphicsComponent.NetworkComponents file.

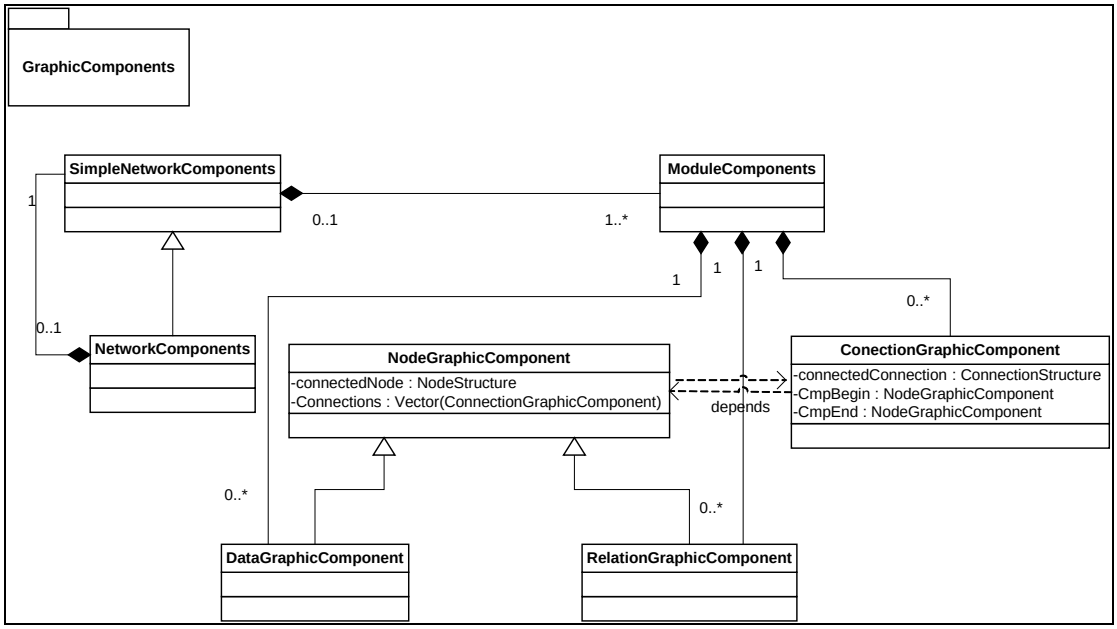


Figure 21 - UML Class Diagram of the graphical part

Concerning the ComponentInfo classes (NetworkComponentInfo, ModuleComponentInfo, etc.), they are used to be saved in a file thanks to the Serializable Java interface.

Forms.MainFormUtils is like a library with a lot of static method useful for graphical processing. The xml package contains all XML processes.

And finally, the Forms package contains all the frames. It is helped by the component_utils package.

Architecture's critics

Considering the architecture of the application, next programmers should be aware of several points:

- As said before, the logical and the graphical parts are separated with no direct interaction between them. This means problems of accessibility and updates : when one part is updated to the other one has to update its repository too. Some fields and methods were built to manage that (*Boolean isNetworkLogicalyModified() and isNetworkGraphicalyModified()*), but they don't seem to be checked in all cases, in every action the user can perform.
- The reference system does not allow a child to know who is his parent (a Node is a child of a Module). Also, entry points to interact with the network are not so easy for new classes.
- The forms (in the Forms package) are not created in the same way. As well in the implementation as in the way the user interacts with them. The authors are talking about the Speedy Design technique. In other words, it is a combination of the View Handler and the Observer Design Patterns. When a component is updated, it tells it to the component handler which knows the

collaboration and the structure between the form and which will ask all required forms to update their view.

F. Requested improvements

Here we expose the asked requirements[8]. Following is the study plan as we received it. Notes that it is not always easy to understand what the authors exactly meant. Plus the authors of this web page were not always known. However, even if the author was present and known, he could not always explain nor translate the requirements.

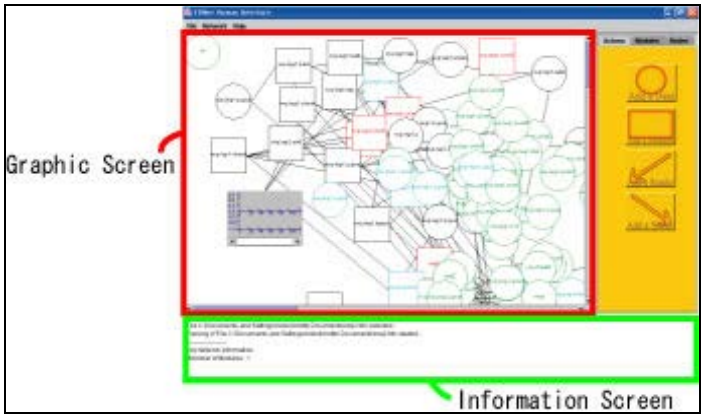


Figure 22 - Human Interface main frame

1. **To indicate the Node information at Information Screen when we edited it.**

When Node is edited, the following information is indicated at Information Screen.

But, indicate only a name with a bold-faced type with a standard type letter except for that.

Necessary information:

Data	Name, host, path, class
Relation	Name, host, path, class
Reader	Name, identical information (information that which Relation read which Data(s))
Writer	Name, identical information (information that which Relation write into Data(s))

2. **To display properties which show information connecting to which Relation (or which Data) on status window when we double-clicked Data (or Relation)(or make the information property window and attach the button displaying it on status window)**

Necessary information:

[8] http://jelly.cs.kobe-u.ac.jp/~tokuda

Data	Relation name which read the Data, Relation name which write into the Data
Relation	Data name which is read by the Relation, Data name which is written by the Relation

3. To patch bug that the lines show Reader and Writer become invisible on HI when we delete Data or Relation on Edit Module window

User Story:

a. This is HI window has read network structure.

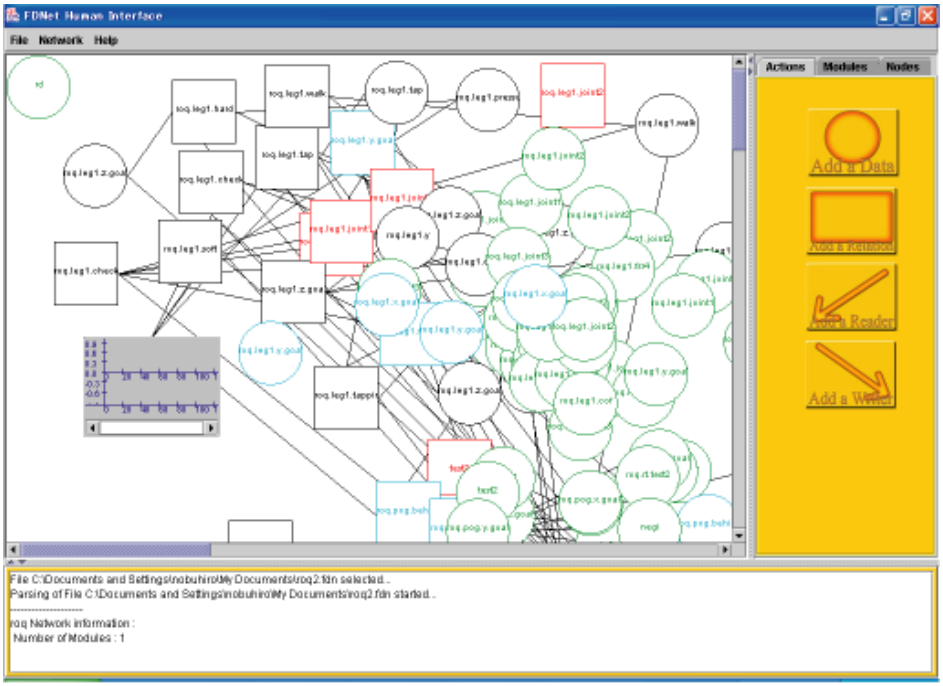


Figure 23 - A loaded Network

b. Open the module information window, Menu -> network -> module information.

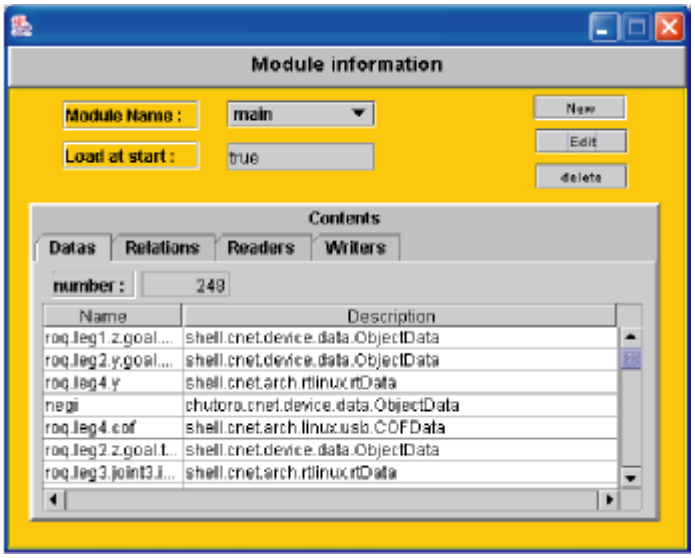


Figure 24 - Module Information Frame

c. Click on the edit button on module information window and Edit Module window is open.

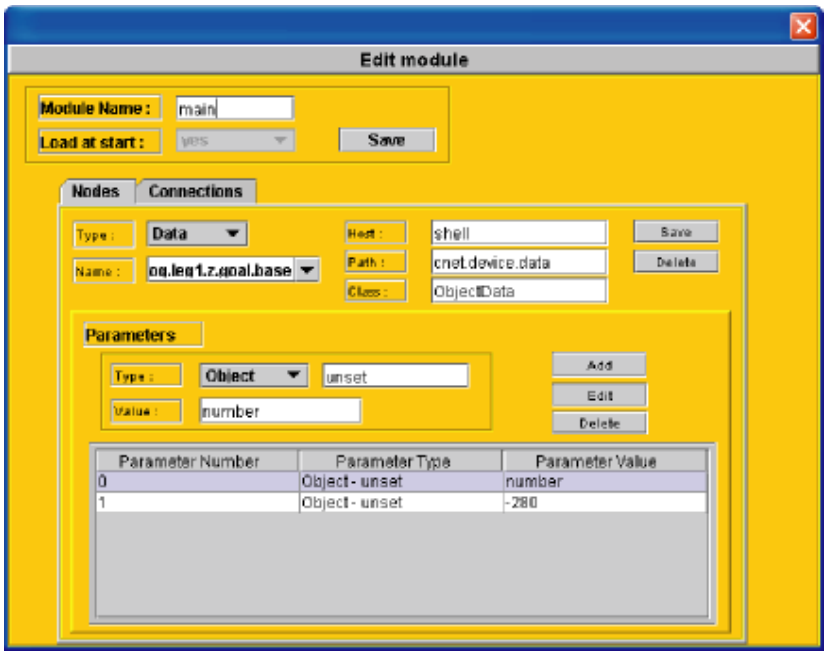


Figure 25 - The Edit Module Frame

d. When we delete Data or Relation, the lines become invisible.

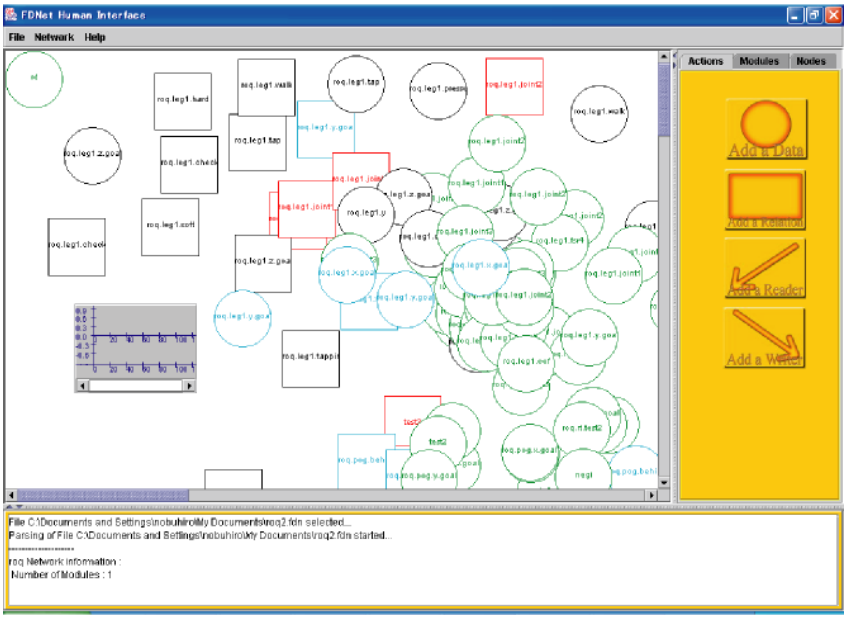


Figure 26 - A bug

The lines Reader and Writer is indicated whatever we may delete Data of Relation on Edit Module window.

4. **To indicate marker of Reader and Writer on HI (attach a mark on Data side of Reader and Relation side of Write)**

   For example, display the marker of Reader and Writer like the following figure.

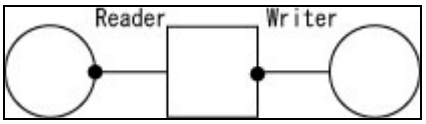


Figure 27 - Marks on Nodes

5. **To choose a .fdn file and a .fdg file separately when Load is selected**

   To use positional information of another file which has been already registered.
   1. To specify a .fdn file and a .fdg file separately when Load is selected
   2. Enable to specify a .fdg file after specifying a .fdn file

6. **To indicate the difference between A.'file which has loaded into HI' and B.'file which is tried to load' (difference: property of Node)**

   1. The 'Comparison' menu to compare the file of A and B is added to the pulldown menu 'Network'.
   2. If that is chosen, it tries to open window that B can be specified.
   3. When B is specified, the difference is indicated in Information Screen.
   4. On the occasion of the designation of the colour:

The information in B which is not in A: blue
The information in A which is not in B: red

   Each Node information is indicated by the above colour.
   But, indicate only the Node name with a bold-faced type and a standard type letter except for that.

7. **Addition of another network structure to network structure which has been loaded already in HI**

   1. The 'Add' menu to add a network structure to one which has been loaded is added to the pulldown menu 'Network'.
   2. The window is open to specify a .fdn file and a .fdg file which it wants to add when Add is chosen.
   3. The properties information of the added node and a skill name is indicated in Information Screen.
   4. Save should have it in the form that network structure is added when saving the network structure.

8. Reference Skill which is loaded in HI

1. The tab of 'Skill' is added to Contents of Module Info window.
2. When the tab of Skill is chosen, the name and the use conditions of Skill which is being loaded at present is indicated.
3. The use conditions: Enable, Disable

    - Enable : the network structure of the skill is indicated in Graphic Screen and we can use the skill.
    - Disable : the network structure of the skill is deleted from Graphic Screen and we can't use the skill.

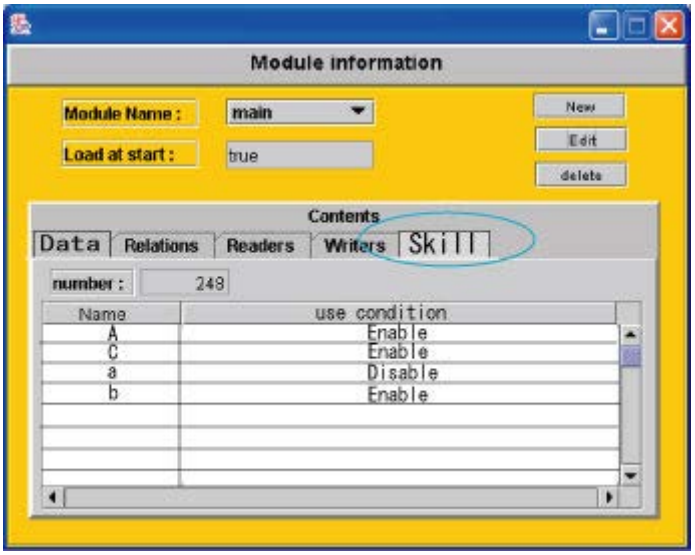


Figure 28 - New requested Modul Information Frame

9. To display by 3-dimensionals Data, Relation, Reader and Writer which are shown by 2-dimensionals at present

Enable to display the information on condition to observe easily

10. To allocate Data and Relation on HI according to groups which are divided into based on regulations for grouping motion skill pattern in network structure file (at present, when HI reads the network structure file, it display all Data and Relation at a single centre)

Enable to display all Data and Relation according to groups which are divided into based on regulations for grouping motion skill pattern in network structure file

11. FDNet-Pattern loading mechanism synchronized with editing on HI

FDNet is the architecture which imitated a human cranial nerves network. Several human skills are expressed in a subnet of the nerve network. As for FDNet, a function which connects each one connection of the nerve network is realized. However, we have some problems in connecting the whole of the sub-net which contains more than one node. After editing sunets in FDNetor, there is a biggest problem when the network can be disconnected and reconnected. This theme

requests to make a structure which it synchronizes the timing when it can be edited from between the robot side and the editor side.

For implementation

* programming language : Java
* development environment : Java2SE V. 1.4.0 or higher version VM and the standard library Java
* version control (CVS) : setting up CVS for HI development

G. Our work

1. Methodology

This section will describe how and why we reached the customers objectives. First the work environment will be presented. Then in regards to this context, a methodology was chosen.

a) Environement

First, we did not have to build a new software and we did not want to change the present architecture of the HI. It was like a prototype that we wanted to improve. We wanted to work together to finish as soon as possible the first objectives of the subjects (improvements 1 to 9) which were smaller than the two last ones (3D representation and FDNet Pattern Loading Mechanism Synchronized with the HI). At last, we were in the same room than the users of this software, thus we could have quick feed-backs.

b) EXtreme Programming

EXtreme Programming (XP) [4] is a software development discipline developed by Kent Beck in 1996. It is based upon four values: communication, simplicity, feedback, and courage. It stresses continual communication between the customer and development team members by having an on-site customer while development progresses. The on-site customer decides what will be built and in what order. It embodies simplicity by continually refactoring code and producing a minimal set of non-code artefacts. Many short releases and continual unit testing are the feedback's mechanisms.

One of the few requirements of XP is to have the customer available. Not only to help the development team, but to be a part of it as well. Every phases of an XP project require communication with the customer, preferably face to face, on site. It is better to simply assign one or more customers to the development team. User Stories are written by the customer, with developers helping, to allow time estimates, and assign priority. The customers help to make sure most of the system's desired functionality is covered by stories.

During the release planning meeting the customer will need to negotiate a selection of user stories to be included in each scheduled release. The timing of the release may need to be negotiated as well. The customers must make the decisions affecting their business goals. A release planning meeting is used to define small incremental releases to allow functionality to be released to the customer early. This allows the customers to try the system earlier and gives the developers feedback sooner.

Here is how we made these small iterations of 1 – 2 weeks.

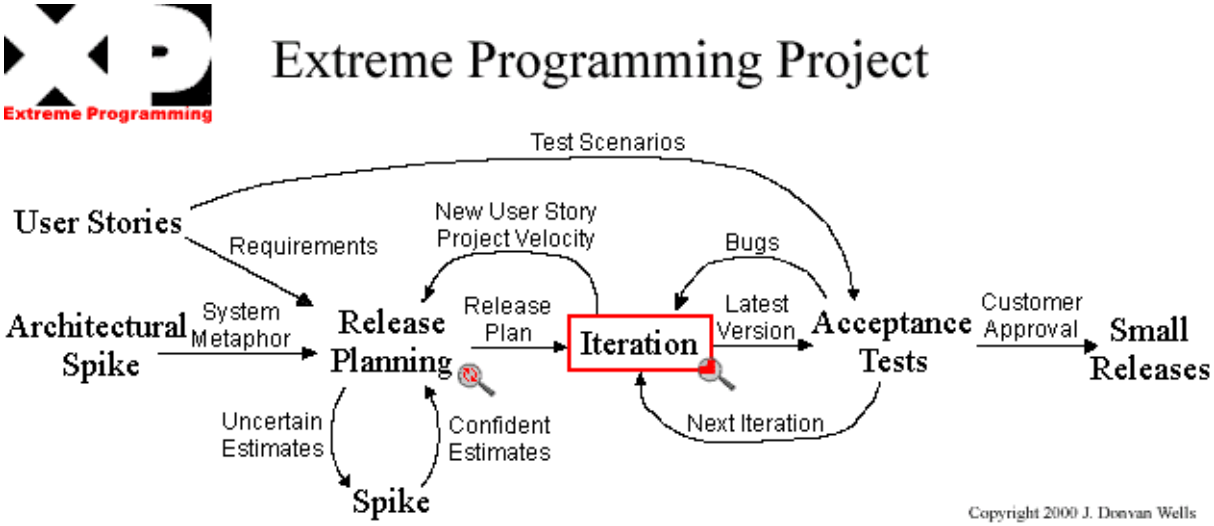


Figure 29 - XP schedule

User stories

Via the Planning Game with the students, XP delegates to customers the authority to define all the requirements with User Stories, handwritten use cases, and to set the priority for those stories. XP also delegates to customers the responsibility to set priorities in such a way as to deliver maximum business value within the resource constraints of the project, and to define tests that will determine when quality requirements are met. XP delegates to developers the authority to estimate, for each story, how long it will take to implement.

Costs Estimation

Their objective is to provide a scalable unit allowing them to value the implementation times of each scenario. These points are called XP Units.

Spike solutions

Explore each potential solution and think about technique and design problems that could occurs.

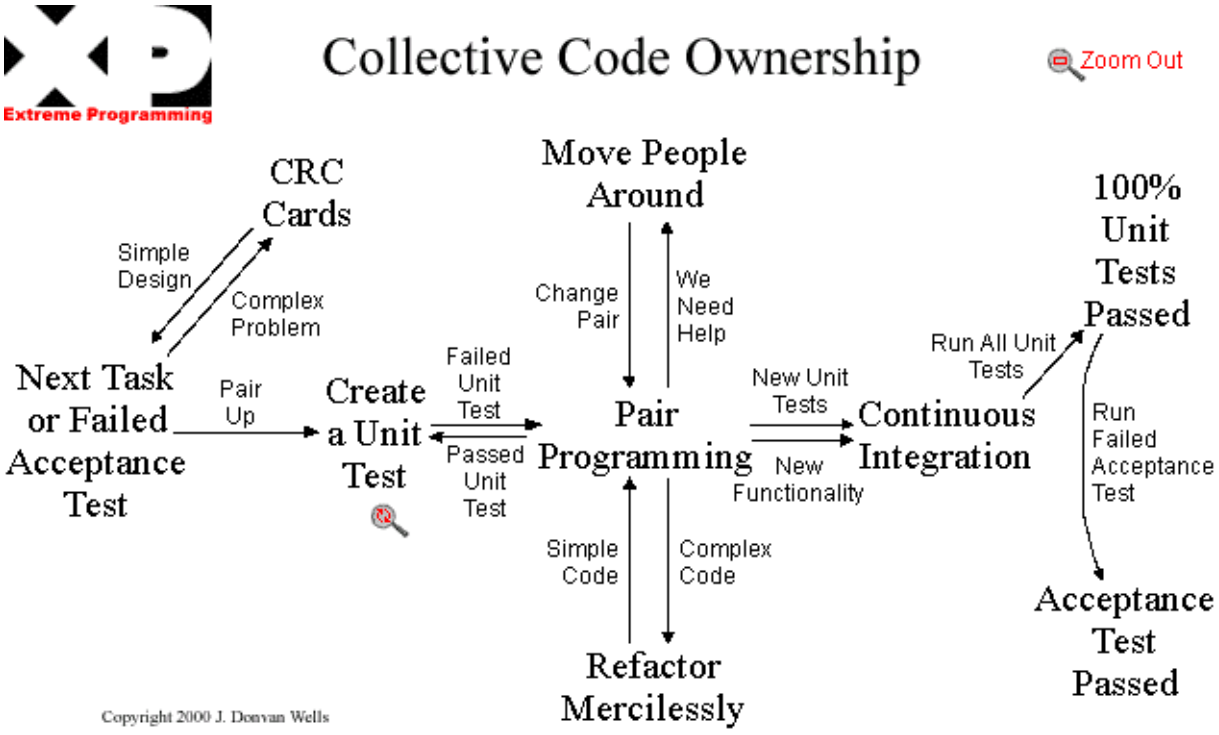


Figure 30 - Collective Code

We choose then to follow the eXtreme Programming methodology. In our case, communication is easy because we are on the same site than the HI's customer. "Beware though, this seems like a long time to keep the customer hanging and the customer's department is liable to try passing off a trainee as an expert. You need the expert." [4] But we had no experts available. The "clients" were not always able to answer our questions. Simplicity of the code and the design was implied because we based on the existing beta software. We were also able to provide regular deliveries and to build unit and functional test rigs easily approved by an engineer student. And we got a lot of courage because we want to solve problems as soon as they appear. As XP suggests, we programmed by pair. It effectively increased software quality without impacting time to deliver.

XP also delegates to developers the responsibility to meet those estimates if at all possible, to measure their performance against their estimates, and to improve the accuracy of estimates over the course of the project. It was what we needed mostly at the beginning of the project and of the new team composed by both of us.

The testing methodology consists of writing the tests before beginning to code (unit test, functional and acceptance tests). The tests will serve as specification support. The unit tests are integrated to the development environment. Junit©, a regression testing framework was a technical solution to automatically manage repeatable tests. But of course, testing a Graphical Human Interface requires also a manual part (for acceptance tests, the human intervention is compulsory).

The system is stored on a shared place. Using a Concurrent Versions System with all other developing teams will make the integration and the building easier. The code must be formatted to match coding standards. Coding standards keeps the

code consistent and make it easy for the entire team to read and refactor the software. Using Java® language, the SUN© code convention has been chosen.

Whatever the methodology, to improve an existing application, it is needed to analyse it well at first. Even though XP advises to get into the work as soon as possible, we spent two months to analyse the application the best we could.

We first read ‘FDNet Human Interface specification’ (by Mr Jadoulle), ‘The report for a meeting’ (by Koji, Yosihisa) and two unsigned documents about the FDNet network Stat Logger. Then we read the thesis of Mr Achbany and Mr Jadoulle[2]. We attached more importance on the HI part.

We were only then able to begin to read the source code. We started with the ‘mainfiles’ package. Therefore we went into the ‘baseObjects’, ‘component_utils’, ‘fdnetconnection’, ‘fdnetutils’, and ‘Forms’and’ GraphicsComponents’ packages. The inside code is very well documented but it took us more time to understand the logical architecture of the application which is less documented.

2. Our solutions

1. Display the Node information on the Information Screen when it is selected.

In fact we did not understand this point very well because it seems that when you edit a Node all these information is well shown on the screen (in the prompted window where you can edit a Node). We added a new function to do that. We improved the interaction with the Information Screen and added some features (like the permit to write in Bold or not).
After talking with the students, we decided to display this information on the Information Screen when you simple click on a node.
In addition, as you can see on the following figure, the selected node is painted in its own color (not only the extremities).

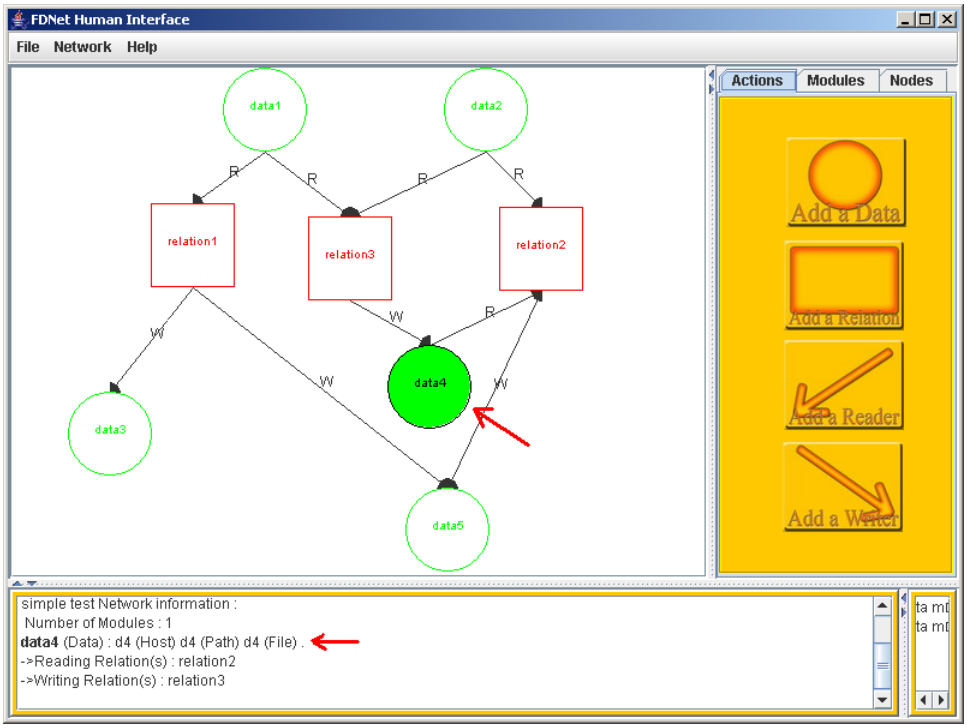


Figure 31 - Selected Node feedback

2. Display the connecting properties of the Node in the Information Screen.

We display the information in the Information Screen in the specified form:

"**RelationName**"(Relation) : "Hostname" (Host) "PathName" (Path) "FileName" (File) .

->Read Data(s): "ReaderName" "ReaderName" … or none

->Written Data(s): “WritterName” “WritterName” … or none

We add other information: the type, the Host, the Path and the File of the Node.

For example:

roq.leg1.hard (Relation) : shell (Host) cnet.arch.RoQ.hard (Path) Hard (File) .
->Read Data(s): roq.leg1.pressure roq.leg1.z.goal roq.power
-> Written Data(s): roq.leg1.hard

IRSI estimates that a simple click is sufficient to display the information. A double-click opens the edit windows and gives you the opportunity to change the information and add parameters.

3. Convert a bug related to the display of the Reader/Writer lines on the Graphic Screen.

We took more time as expected to resolve this point that we thought because it is difficult to understand a so huge code you did not write by yourself and for which you cannot find design documentation (like UML diagrams: sequence, interaction, class, or persistence diagram). It appears to be quite dangerous because it was a question of synchronization with a graphical and a logical change in their respective brain/mind.

By the way, this is the only bug asked to convert by IRSI but others bugs (of the previous version) were found during our work. We tried to fix as much bugs as possible even if it was not required. Finally we spent as much time on the non-required fixing as we did for the requested improvements, but it was necessary.

4. **Show a mark attached to the Connection lines in the Graphic Screen.**

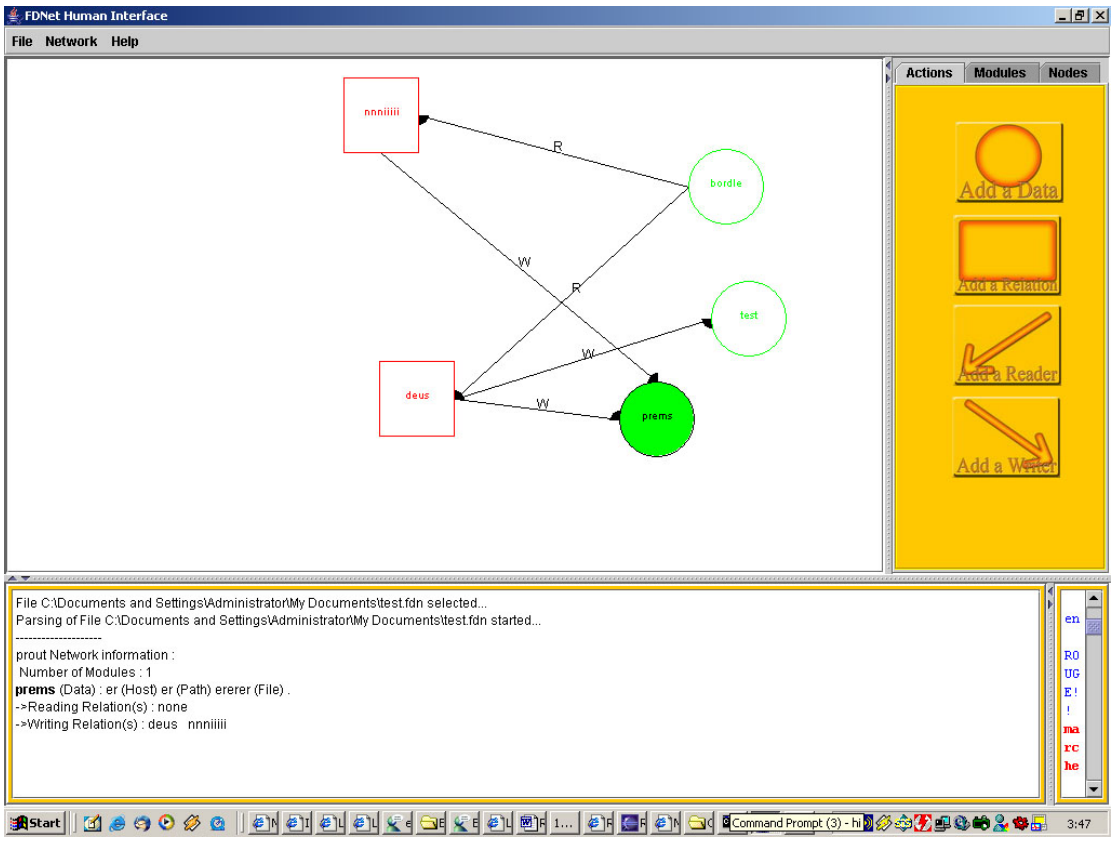


Figure 32 - Connection's Mark

As shown we added a letter (R, W) to simplify the visualization because several connections on the same relation make it difficult to discern quickly which connection is a reader or a writer.

We had many problems to display a triangle on the connection so the form is a quarter of a circle stuck to the element.

5. **Enable to choose a specific .fdg (graphical) file for the current loaded network.**

We first added the possibility to load a different graphical file. But after this had been done, we though that the link between the .fdg (the graphical file – containing the elements position…) and the .fdn (the logical file – containing the elements, the elements properties…) should be stopped. For example, what would happen if you load another graphical file and then click on the 'Save' button (which saves both the logical and graphical file)? That is why we added the possibility to save only the .fdg or the .fdn file.

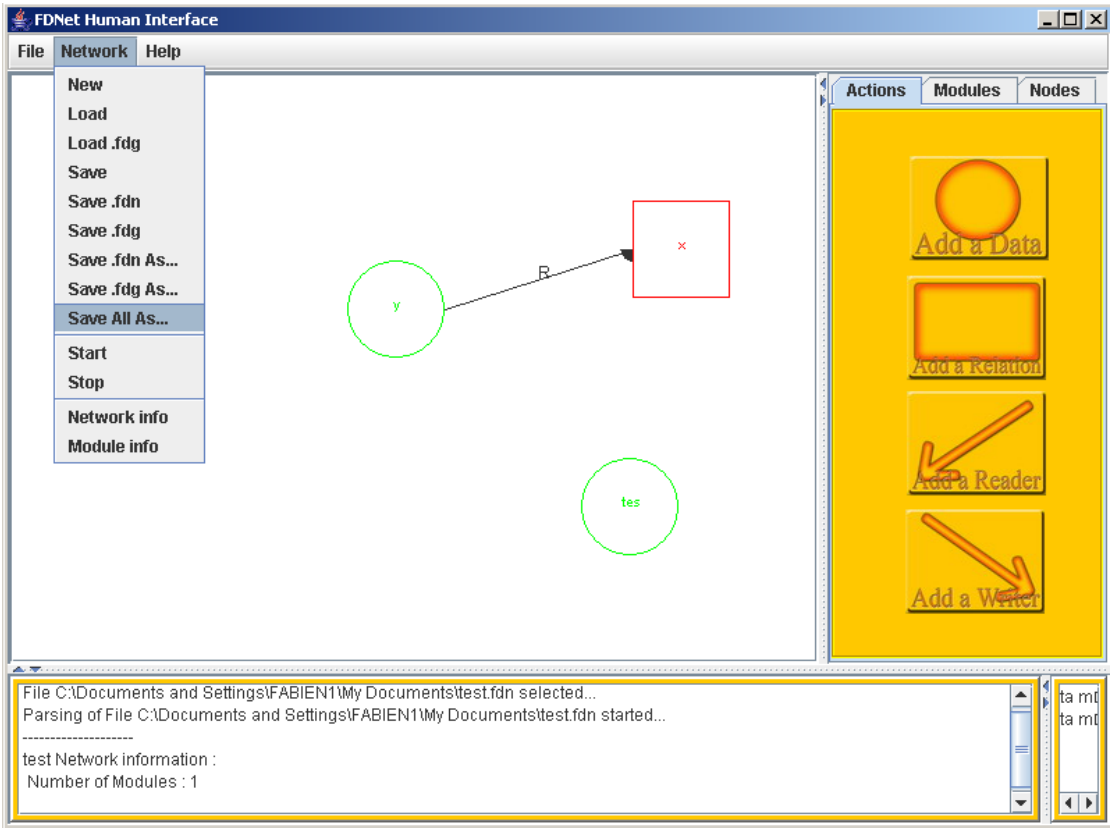


Figure 33 - The new Network Menu

6. Compare two logical networks

The aim is to be able to compare two networks, one already loaded and a second one chosen by the user. The principal difficulty was to take care about the changes the user could have done. So we chose to save the loaded file in a temporary file just before the comparison. The result is shown as required in color in the comparative screen as shown in the Figure 34 – The Comparative screen.

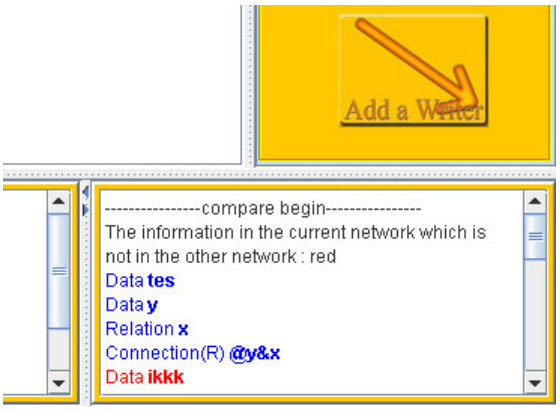


Figure 34 – The Comparative screen

7. Add a logical network

The main problem encountered in this improvement was to separate the logical and graphical save and load methods. Some other problems came out because different users were not agreeing on the rules to follow in order to resolve the conflicts that appear. For example: what to do if a node already exists?

8. Manage Skills

By "**Skill**", we imply a set of nodes and connections that can be enabled or disabled dynamically once the network is started. A disabled skill is invisible on the screen and cannot interact with the network. A node belongs to an enabled skill and another disabled one at the same time stays on the screen and is considered as enabled.

- **Enable a Skill:**
    - All the Relations belonging to the skill are shown on the Graphic Screen and accessible. All the Datas directly connected to these relations are shown and accessible too.
    - When enabling a Skill, if a component already exists, it will not be overwritten (the disabled component will be lost).
- **Disable a Skill:**
    - All the Relations belonging to the skill and for which all their Skills are disabled are hidden of the Graphic Screen and not accessible anymore. All the Datas directly connected to these relations and for which all their directly connected Relations are hidden will be hidden too and not accessible anymore.
    - It is still possible to access a disabled Relation in the frame of adding and removing a Relation to/from a Skill. They are shown in a red colour.

Here, it was very difficult to make the difference between a module and a skill (the difference is that nodes or connections can belong to different skills but only to one module). But that is not all; we did not receive any example of skills usage. So we only made what the students asked for: add/remove relations in skill and enable/disable skills.

Because of the Skill concept we had to adapt the software's design. When a Component was disabled because of a Skill, the component was moved into the Simple part (SimpleNetworkInfo and SimpleNetworkComponents) of the network. The components located in these two networks are neither accessible, neither visible.

Following is the IRSI's answer of the questions: "How will you use a skill? What about a module? What are according to you the differences between the both of them?"

The answer is: "How to use module is on the definition.

At present, because we make a foot program independently,
Each network is made to cope with each skill.
This is a way to divide into some modules,
and some modules combins each other into large network.

In future, we hope to distinguish modules from the skills,
because one datum is depend on some modules at one time.
We had better carry such as the data away from modules."

H. Evaluation and comparison of the FDNet's HI

In this section the FDNet HI and 3 other HI will be analyzed through a cognitive walkthrough and a think aloud evaluation. After this, the developed HI will be compared to the others and criticized. And finally, some improvements will be proposed.

1. How to evaluate a HI

The evaluation of a HI starts at the beginning of the development. When an objective is determined by the client, programmers have to discuss with him to select some design's alternatives in the HI. Then the work may begin. During this time the user has to be requested to test the prototype and gives his agreement to continue on this way or to change some parts.

Finally when the client accepts the HI, some tests have to be done on common users to see if the software is efficient on average people. We have two kinds of evaluation: one with a user and another one without anyone.

In the first one, the attention is focused on the user; he may be observed during the test or questioned just after. For example the users could be in an equipped room where everything they are doing or saying is monitored. Cheaper alternatives are observers (hidden or not) watching the users in the same room and a think aloud evaluation could be performed.

The common point in all of these methods is that the user will be questioned about different aspects of the evaluation as well as the software and the last part is to consider what the user is able to do or not and extract figures of the user's performances.

In the second one, the evaluation without the user, the HI will be tested by experts, who will check if some ergonomic heuristics are respected and estimate the software behavior with models, a heuristic evaluation or a cognitive walkthrough (CW).

The evaluation without user is separated in two groups: method based on task (this method forces the observer to monitor the different actions and reactions of the user while doing a requested task), and method not based on task.

In the first group we have methods like GOMS (**G**oals, **O**perators, **M**ethods, and **S**election Rules), KLM (**K**eystroke-**L**evel **M**odel) or the cognitive walkthrough. And in the second one are heuristic evaluation ([NIELSEN 93]) and evaluation with ergonomic recommendations (the 8 design points [FBODART]).

2. Evaluation of other HI and comparison

In this section, some HI will be evaluated and compared to the FDNet HI. These evaluations will be an "a priori" evaluation (without users, only with experts) and subjective; after this, a think aloud evaluation will be made, only on the FDNet HI, to provide a view more similar to the user's point of view.

These two evaluations are complementary and will help us to provide a point as similar as possible to the users' one.

a) The "a priori" evaluation:

First, how to perform a cognitive walkthrough will be described. Because of the several scenarios and possibilities and because we only focus on the resemblances with the FDNet's HI and manipulation of "nodes" and "connections", the CW will also be simplified.

"Choosing the right tasks to examine is key, since aspects of the interface not involved in the tasks that are chosen will not be examined" [*LEWIS97*].

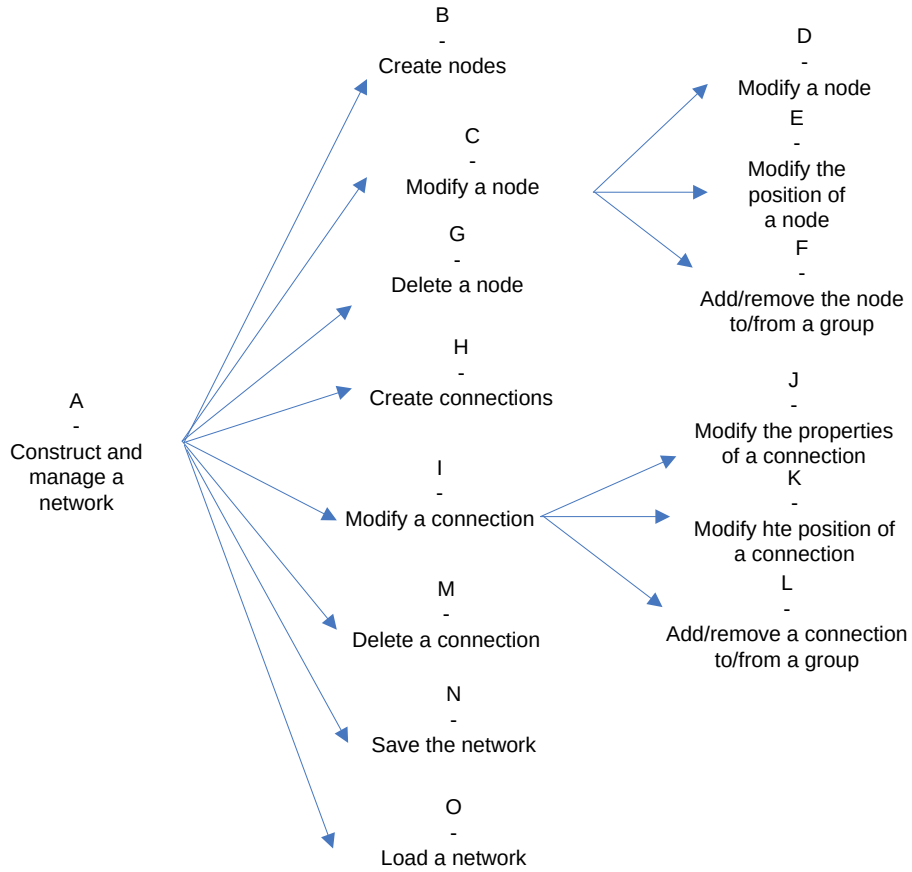
CW and its three phases will shortly be explained but first the environment of a CW must be described.

Obviously a HI is needed or at least a prototype and, with this HI, a task to analyze has to be defined. But this is not sufficient; we also need a usability context (user stereotype and work tools), utility and usability criteria and a structured goal to be able to construct the scenario.

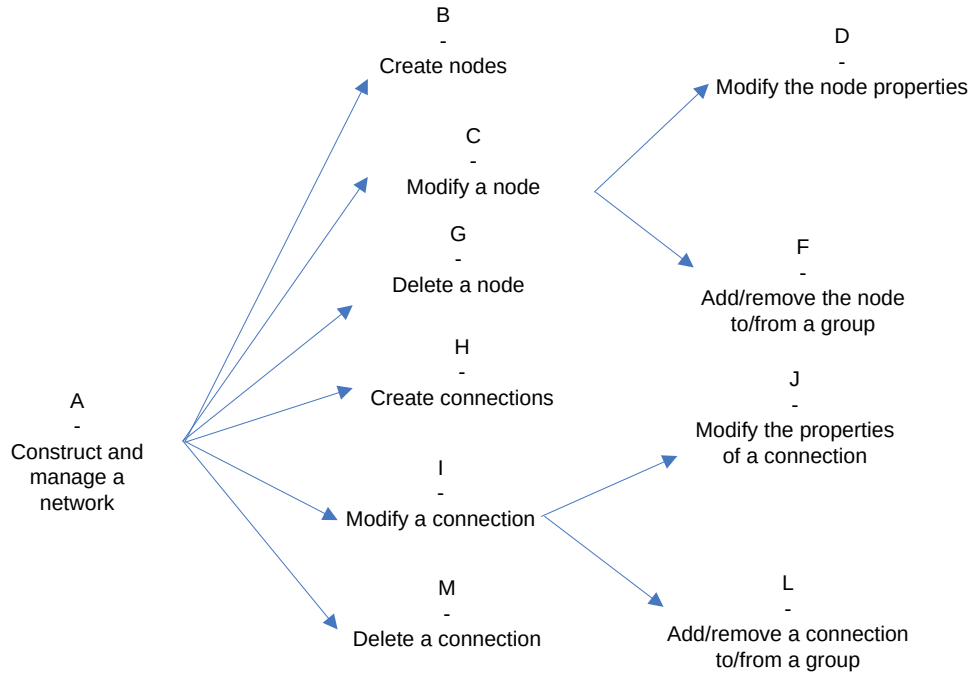
(1) The scenario

Phase 1: Definition of the user's way to use the HI

First, the main tasks could be defined like this:



But as said before, the CW will be simplified by deleting some tasks:



Then the experts have to answer the seven following questions for each final under-goal (2) and for the actions (5), those will be answered in relation with all the screen-shots:

Q1: will the user accomplish this under-goal to reach the main goal?
Q2: what does the user know to accomplish the under-goal and does he have it?
Q3: does the user understand that the task is available to him?
Q4: does the user associate the under-goals with the actions he's performing?
Q5: does the user "catch" the feedback?
Q6: does the user understand the feedback?
Q7: does the user know that he's progressing in the main task?

Secondly, all the last sub-goals (not c for example) have to be described like this:

Task name	User's action	Software's feedback	Comment
B	Click on the button	The node is created	Trying to describe what the software did (with screenshots if possible)
C	…	…	…

Phase 2:

For each task, questions have to be answered to determine the gap between the physical environment and the psychological variables [NORMAN86]. We have two different kinds of questions: questions related to sub-goals and questions related to actions.

For all the sub-goals we have to look if the user will accomplish all the goals belonging to them. For example, if the user wants to add a node in a group, he has to create the node and then to add it to the group. “Create the node” can be seen as a sub-sub-goal. If goals are too hidden behind a lot of sub-sub-…-goals the user may feel that the action is impossible to do.

To be sure that the user may find a way to achieve his aim, he could be helped by the HI or the HI could be build like software the user already knows.

This introduces the experience of the user and what he has to know to use efficiently the HI, plus a second question: does the user know how to solve a problem or how to accomplish a task?

For the actions we have five typical questions: (1) does the user understand that the task is available to him?; (2) does the user associate the sub-goals with the actions he’s performing?; (3) does the user “catch” the feedback?; (4) does the user understand the feedback?; (5) does the user know that he’s progressing in the main task?; (more details are available at http://www.hec.unil.ch/fbodart/IHM/Chap7/cogn_questions_action.html).

Phase 3:

In the last phase user’s answers will be analyzed and problems will be underlined. The result will be shown in a table where the problems will be present in relation with the utility and usability criteria.

(2) Benefits and limitation of a CW

Benefits are that we do not use users and the experts are forced to focus only on the main tasks. But the experts have to know the users’ characteristics and their background. Others problems are that this method is really expensive in time and worst is that this method is based on the capacity of the experts and on their capabilities to simulate the future users.

(3) Evaluation of Simula® [SIMULA]

This software was created for mathematicians and computer scientists to manage Petri networks, logical circuit or schematized information flows. It can simulate results to detect some behavior. In this HI we focused on the Petri nets options because this is the most alike FDNet HI.

Usability context: the user’s profile

The designers assume that the users will be mathematicians or computer scientists. So the software does not explain how to construct a correct network. The user is supposed to manage Petri’s nodes and transitions at least on paper. On the other hand the users are not supposed to create networks bigger than 50 nodes even it is possible.

Utility and usability criteria

The main criteria are the execution speed and the error rate. Because the time is taken to create the semantic, there is no need to waste time in creating elements. And secondly errors in the HI are errors in the network.

Through this evaluation (available in the appendices) some problems are discovered: the creation of a connection requires the use of the ALT key and the mouse at the same time, this is not really common. The user expects to see an item connection in the Tools panel, plus this ALT key is the only way to create a connection. Fortunately the user may read a permanent written hint at the top of the screen (“Hint: In Petri-Nets, to drag Arc from Place to Transition, Press ALT key, then click near the edge of the place or the transition”). But this is not sufficient.

Secondly, to select a connection the user has to click on it. There is no combo box as it is for the nodes. Then, in big networks or when several connections are connected to a same node, it can be difficult to select the right connection.

And finally, to change the properties of an element the user has to press RETURN after a change, also when the value is in an incremental box as we can see on the Figure 35 - Non sensitive box.

Figure 35 - Non sensitive box

On the other hand the idea to use the ALT key to create a connection and making available a keyboard shortcut is a good point. Plus the user may be accustomed with the CTRL key to duplicate elements. So it will not confuse him too much.
The details are available in the annexes.

(4) Evaluation of HPSim® [HPSIM]

HPSim was built to manage Petri nets and simulate them. So the usability context and the usability and utility criteria are the same as for Simula. Plus the two HI are very similar but the important point in this evaluation is the way to construct elements. The main difference is that the user has to switch the mouse pointer into constructors with the mode panel. No drag and drop are available to create element

This kind of HI is better if the user knows how to use it and if he knows the network he wants to construct before too. For example, he can create all the nodes and their properties and then connect them. This constructor mode is better to make a “copy” of a paper version of the network that the user has drawn before.

In the evaluation available in the annexes two problems are shown. The first appears, as said, when the user wants to switch from mode. For example if the user wants only to create a single node and then a connection, he may forget to change the mode and construct more nodes. Especially, as shown on Figure 36 – HPSim Handler when he wants to select an element, transforming the mouse into a pointer could be strange for him.
Another problem is when the user wants to delete a node; then the orphan connections will disappear and the user can be surprised, he may lose information in the connection. The software tries to help the user since it keeps the syntax (a connection cannot stay connected only to one node) of the network, but some semantic may be lost (the properties of the connection are deleted too). The user will be forced to rebuild the lost connections. But he can learn now to change the properties of a node instead of deleting it and then reconstructing it.
HPSim® was designed for mouse's use, there is no keyboard shortcuts (only CTRL + X, C and V), nor menu's ones.

Figure 36 – HPSim Handler

However, the constructor mode is better to construct well known network and the HI provides a zoom to focus on different parts of the network.
A survey statistic made in March and April 2002 indicates "app. 75% of the user's rate simulator and editor from good to excellent" [HPSim eval]. Final users are satisfied with this kind of HI once they are accustomed to it.

(5) Evaluation of Visio® [VISIO]

Usability context:

The users are users of a Microsoft Windows® environment and they are maybe users of the Office® suite. So using a computer is not new and, most likely, the users know how to construct a drawing with lots of entities with an HI. Visio® was created for the average computer's user and not only for scientists, so the HI was designed to be as simple as possible. That is why Visio® is a good quality work and source of inspiration.

A problem is that the 'Group' and 'Ungroup' buttons are in the 'Shape' menu. It is maybe a little difficult to find these options. The shape word doesn't make the user thinks about grouping nodes.

Anyways Visio® 2003 enables easy assembly of diagrams through dragging predefined Microsoft Smart Shapes® symbols coupled with powerful search capabilities to locate the right shape, whether it is on a computer or the Web. The tool is quite complete with lots of practical functions. The graphical panel (the one in the centre) is squared and the technique of Connection Points to glue objects is very useful. The interface is clear and a lot of feed-backs are given. The zoom function is available and also the CTRL way to duplicate.

(6) Evaluation of FDNet's HI:

Usability context:

This software is dedicated to engineers and electronics specialist. They are accustomed to computers and know how to construct a logical network. The only main difference with the tools of previous sections is that the network will work in the real world and may damage the robot.

Utility and usability criteria:

The learning time:

The learning time is not so important because user may create the network only on the HI and not running the network with the robot.

But a too long learning time may discourage the user to learn how to use the HI.

The speed of execution:

This is important because of the number of nodes to create. The time is mostly allowed to create a workable network. There is no need to waste time in creating elements.

Error rate:

Errors are directly influencing the network, so the error rate has to be as lowest as possible.

Remanence period:

It is not so important because the design of the elements is based on the convention in the FDNetwork then the cognitive work of the user will be focalized on the way to draw the elements like he does on paper. Moreover he uses to work in this kind of environment and know others HI designed on the same way.

But not natural use of menu can be dangerous and make the user feel uncorftable.

Satisfaction of the user:

This is the most important criteria because this is the user point of view.

The structured tasks:

Task name	User's action	Software's feedback	Comment
B	Click on the "add a data" or on the "add relation" button	The node is created and the properties panel is automatically open	The node is created under the mouse [figure-FDNet 1]
D	The user can double click on the node or go to the Network menu choose the right module (the one that contains the needed node)	The properties are updated	A panel appears and tell the user that the properties are updated [figures- FDNet 2,

	and click on edit, then choose the right node with his name and change the properties and click on save		3 and 4]
F	The user has to right click on the node and then on skill or go in the Network menu then choose the Skill info menu and then click on add relation and choose one	Nodes are grouped	The relation is added in the skill window [figure-FDNet 5 and 6]
G	There is only one way to delete node: the user has to know the name of the node and which module it belongs to. Then he has to go on the Network menu choose module info, select the module, click Edit, choose the node and click on delete	The node is deleted	The node disappears and the orphan connections disappear [figure-FDNet 7]
H	The user clicks on “Add reader” or “Add writer”	The connection is created	A connection is created between the two specified nodes (the connection got a direction and a type) [figure-FDNet 8]
J	This is the same as for the nodes but the right click is not available	The properties are updated	A panel appears and tells the user that the properties are updated [figure-FDNet 9]
L	The connections can’t be grouped.	/	/
M	See G.	The connection is deleted	The connection disappears but the nodes stay [figure-FDNet 9]

Figure - FDNet 1

B task (create a node):
Q1: Yes, because of his experience of creating networks, the user knows that he must create nodes.
Q2: Yes, the user will understand because of his experience of the system.
Q3: Yes, the user will understand because of his experience of the system. Plus under the button ha can find Add a Data/Relation written.
Q4: Yes, because of his experience of such possibility in the HI.
Q5: Yes the user will see the feedback because the properties panel will be open to initialize the properties.
Q6: Yes the user will understand the feedback because of his experience. Plus a panel will appear to warn him that the node has been created.
Q7: Yes because he knows that constructing nodes are the base of a network because of his job.

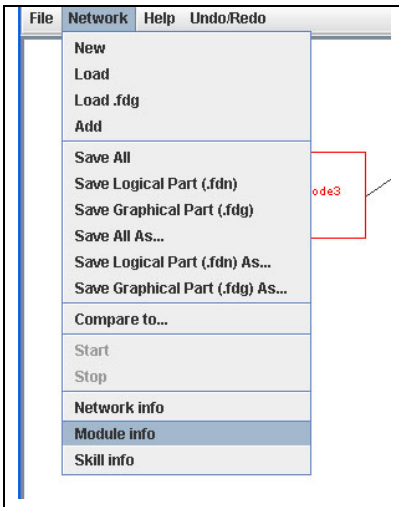


Figure - FDNet 2

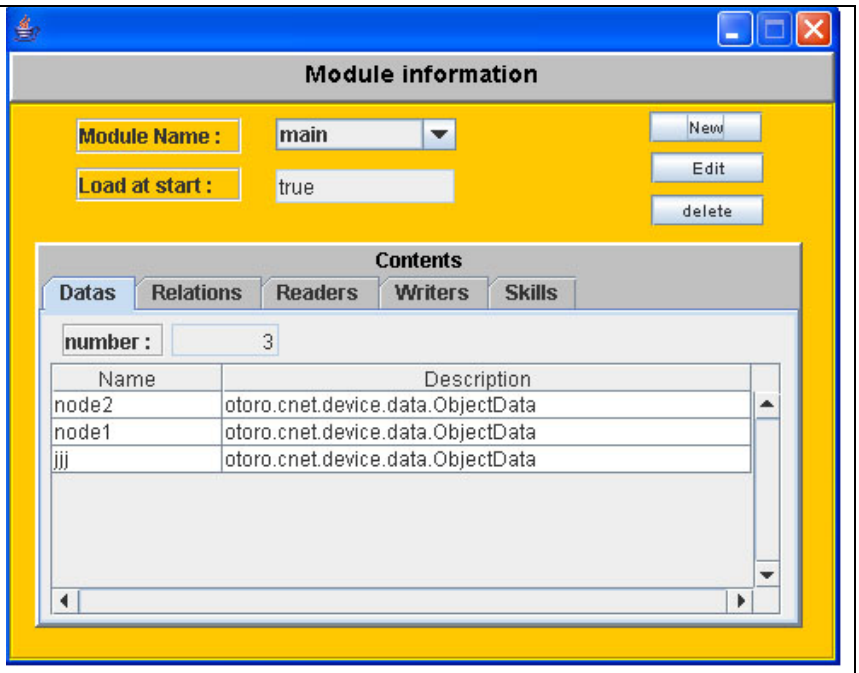


Figure - FDNet 3

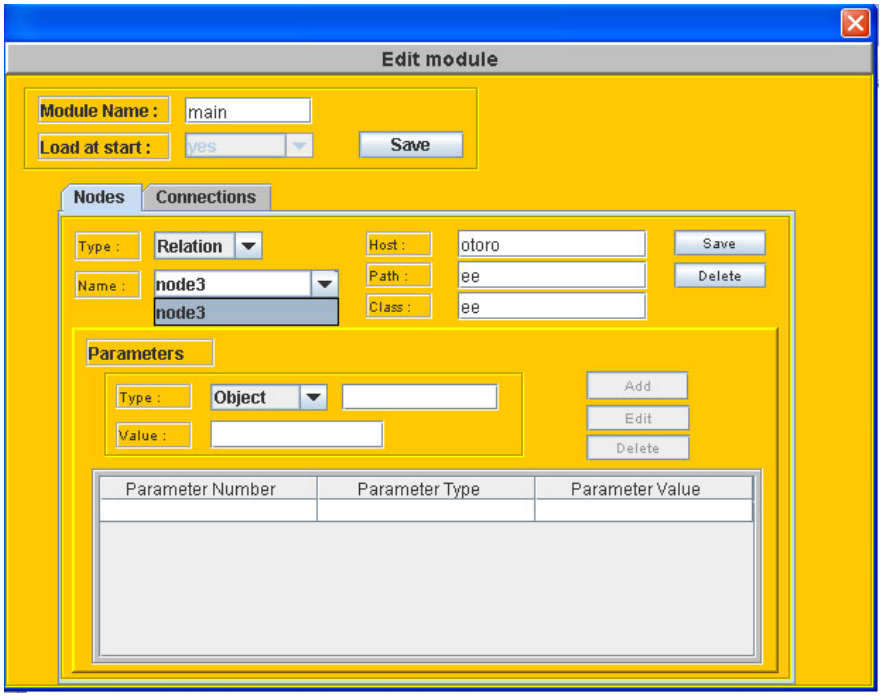


Figure - FDNet 4

D task: (change the node's properties):
Q1: Yes, the user will try to solve this sub-goal because he knows that nodes posses properties.
Q2: Yes, because of his experience of the system.
Q3: Not necessary because he has to go in several windows to access the node's properties, but he can double click on the node and access to the properties.
Q4: Yes because the name of the window is Edit "name of the property".

Q5: Yes because the value will change where he is typing and the node on the main panel may change, but he has to click on save.
Q6: Yes because he knows the FDNet's conventions
Q7: Yes because he knows that constructing nodes are the base of a network because of his job.

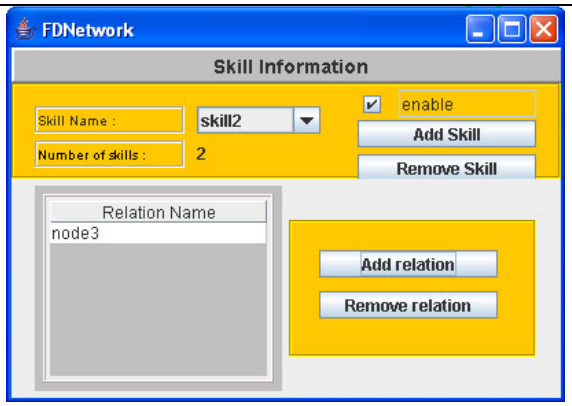


Figure - FDNet 5

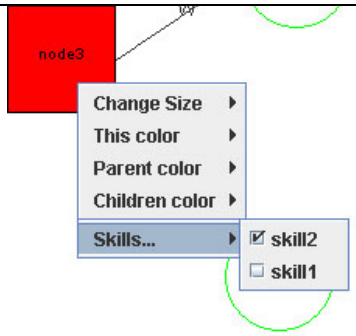


Figure - FDNet 6

F task: (group nodes):
Q1: Yes the user will try to solve this sub-goal because he knows that has to group nodes.
Q2: Yes, because of his experience of the system.
Q3: Yes, the skill menu exists and his implemented in the right click.
Q4: Yes, the user knows the FDNet's conventions and the skill term.
Q5: Not necessary, if the user use the right click there is no feedback like in the skill window, the color is not directly changed or asked. The user manages the colors separately.
Q6: Yes because he knows the FDNet's conventions.
Q7: Yes because he knows that grouping nodes may be useful in a network.

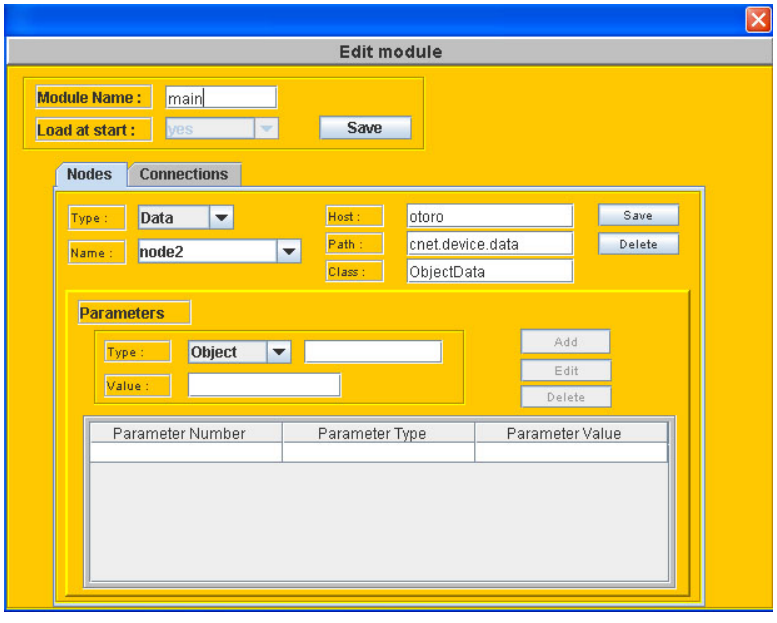


Figure - FDNet 7

G task (delete a node):
Q1: Yes, because of his experience is such network, the user knows that deleting nodes is useful to manage errors.
Q2: Not necessary, because there is no edit menu and the right click doesn't provide the delete function. As describe in the structured tasks the way to achieve the G task is really complicated.
Q3: Not necessary, nothing is clearly exposed as a delete function.
Q4: Yes, he already did this kind of operation in other HI.
Q5: Yes because his attention is focalized on the node and the HI will warn the user if he really want to delete the node.
Q6: Yes, because of his experience he will understand what's happened.
Q7: Yes the use will see that he's progressing in the task because of his experience Petri nets and systems.

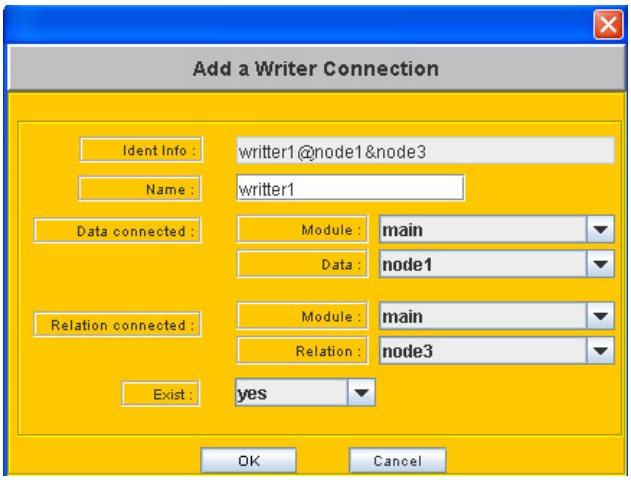


Figure - FDNet 8

H task (create a connection):
Q1: Yes because he knows that he must create connections between the nodes (also it's not a network).
Q2: Yes, but he can be surprised the see directly the Figure – FDNet 8 appear he can't choose himself the nodes on the graphical screen.
Q3: Yes, because of his experience, the user will understand that the library is available since he saw the Handler.
Q4: Yes, because of his experience of such possibility in the HI.
Q5: Yes, the HI will warn the user and tell him that the connection is created but the connection may be created outside of the screen.
Q6: Yes the user will understand the feedback because of his experience.
Q7: Yes because he knows that constructing nodes are the base of a network because of his job.

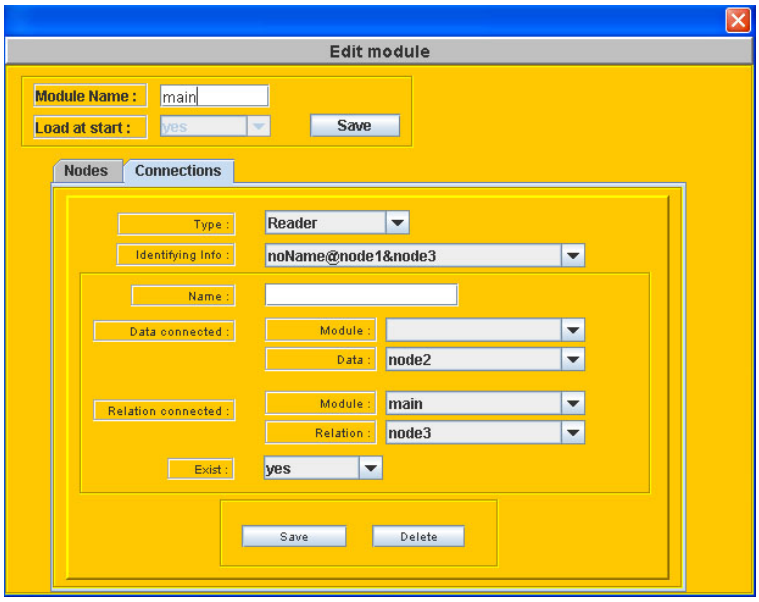


Figure - FDNet 9

J task: (change the node's properties):
Q1: Yes, the user will try to solve this sub-goal because he knows that relation posses properties.
Q2: Not necessary, the menu is hidden by the name of the menu "Network".
Q3: Not necessary because he has to go in several windows to access the connection properties, the user can't select a connection.
Q4: Yes because the name of the window is Edit "name of the property".
Q5: Yes because the value will change where he's typing but he has to click on save.
Q6: Yes because he knows the FDNet's conventions
Q7: Yes because he understand how FDNet works.

M task (delete a connection):
Relate to Figure – FDNet 9 too.
Q1: Yes, because of his experience is such network, the user knows that deleting nodes is useful to manage errors.
Q2: Yes, because of his experience.
Q3: Not necessary, because there is no edit menu and the right click doesn't provide the delete function. The user has to reach the Edit module and choose the right connection (as for properties changes) and then click on delete.
Q4: Yes, he already did this kind of operation in other HI.
Q5: Yes because his attention is focalized on the connection and the HI will warn the user if he really wants to delete the node.
Q6: Yes, because of his experience he will understand what's happened.
Q7: Yes the use will see that he's progressing in the task because of his experience Petri network and systems.

The discovered problems:

Problem to update the properties in the J task	The user has to go in the Network menu choose the right module (the one that contains the needed node) and click on edit then choose the right node with his name and change the properties and click on save. Note that the double click is not available for connections. It's make the number of window and menu to open pretty big.
Cause of the problem	The right click is used for other functions (group, change size, change color…)
Relation with the utility and usability criteria	This is a problem for the learning time, for the remanence and especially for the speed of execution because the user will forget it the next time. It can affect the subjective satisfaction

Problem to access to the right elements to edit them	The problem is that the user has to go to the Network menu and then to the Module info, these two names are not natural for the user.
Cause of the problem	No Edit menu
Relation with the utility and usability criteria	This is a problem for the learning time and for the remanence period

Problem to delete an element	The problem is the same as for the properties' changes because the only way to delete a node is to go in the Module info panel and choose delete
Cause of the problem	No delete action is quickly available
Relation with the utility and usability criteria	This is a problem for the speed of execution and for the remanence period. It can affect the subjective satisfaction

The solution for the delete operation is really simple, the right click could be updated or add an Edit menu in the menu bar. By the way an Edit field could be added in the right click of mouse for the properties' changes. Then it has to be reorganized, the way to use it not efficient.

For the deleting problem, the designers could add it to the Edit menu or into the right click too, but most important is that the user has to be able to select connection with the mouse and meanwhile property field will be accessible.

(7) Benefits and possible usage of the Excentric Labeling in the FDNet HI

For the comfort of the user, research in the Human-computer Interaction Lab at the University of Maryland concluded that “The widespread use of information visualization is hampered by the lack of effective labeling techniques“[EXENTRIC LABEL]. They propose the "excentric labeling", a new dynamic technique to label a neighborhood of objects located around the cursor. This technique does not intrude into the existing interaction, it is not computationally intensive, and is easily applied to several visualization applications. A pilot study indicated a strong speed benefit for tasks that involve the rapid exploration of large numbers of objects. Their method is really simple. If the mouse is not used for at least one second, a red circle appears around the pointer and ‘unpacks’ the nodes (Figure – Excentric labels 1). If the mouse is shacked or moved too quickly, the circle disappears. When the mouse pointer is transformed in the red circle it provides to the user the deployment of the nodes’ labels and the number of nodes in the red circle (Figure – Excentric labels 2). The demonstration is available at http://www.cs.umd.edu/hcil/excentric/dist/bin/test7.html.

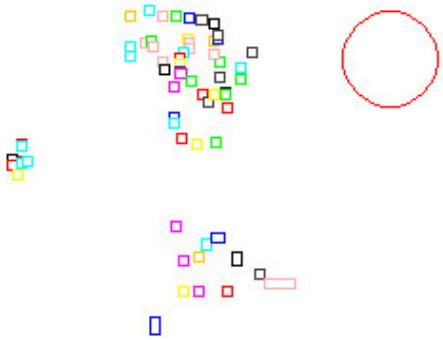


Figure - Excentric labels 1

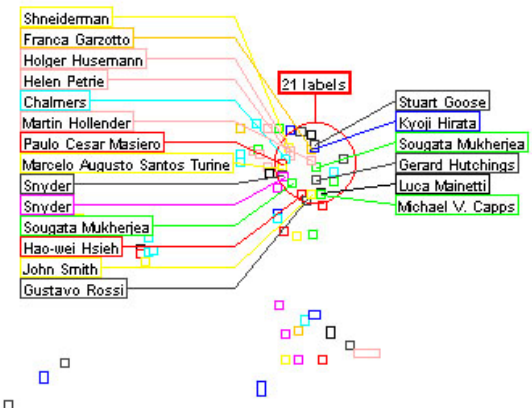


Figure - Excentric labels 2

Of course the combo box in the Module information panel has to stay and make available the easy choice of an element in a huge network where nodes can be superposed on the screen or far away from the point of view. But the explained method may help the user to find the node's name, or to understand a loaded network he did not create.

b) The evaluation with the user

In this section FDNet HI will be evaluated through a subjective evaluation with the users.

Thinking aloud is a very efficient way of getting a lot of qualitative data from a user. As the name suggests, the user should think aloud while performing some specified task with the system. By verbalizing those thoughts, test users enable us to understand how they view the computer system.
The user usually gets a task to perform and is asked to think aloud while doing this. The experimenter often need to prompt the user to think out loud by asking questions like "What are you thinking now?", "What do you think this message means?", "Why do you do this ? (to know the user objectives), "How do you do it?" (to gather information about the sub-tasks and then being able to ask questions about those), "Why do you not do this in the following manner?" (to know the user's choices), "What are the preconditions for doing this?" (to evaluate if the user understand the conditions and the workflow in the HI for this task)…
The experimenters are not supposed to answer any questions or draw the attention of the user to certain aspects of the interface that the user is not clearly working with.

The results from a thinking aloud session are a lot of qualitative data. Due to this fact, the number of users does not have to be so large, a lot of important and valuable information could be obtained with just a few users.

(1) How to Perform a Thinking Aloud session

1. The user is instructed how a thinking aloud session should be performed. This could either be done by showing a videotape with a session that shows how the user is supposed to do when interacting with the system, or by having the experimenter showing how to do.
2. The experimenter informs the user about the task that is to be performed and the system on which this should be done.
3. The session starts. The user thinks out loud while interacting and the experimenter prompts when the user is silent.

(2) Benefits and Limitations

A lot of qualitative data can be collected from only a small number of users. This data often contains explicit quotes that could be used to make the test report more readable and memorable.

Since the user thinks aloud while interacting, the experimenter gets a very direct understanding of what parts of the dialogue that causes the most problems.

Additionally, a lot of problems that the user would not remember in an ordinary interview shows in a thinking aloud session.

"The main limitation is that it seems unnatural to most users to think aloud when using a system and the tasks could feel harder to perform due to this. A different case study has also showed that the verbalization could lead to an increase in performance from the test users". [Berry and Broadbent 1990]

Another important limitation is that the method requires a lot of interpretation from the experimenter. A lot of user "theories" about what caused the different problems will be presented by the user, and the experimenter should not give too much weight to these theories. The user is an expert on telling what he or she does, not on interpreting why.

(3) The evaluation

In the think aloud evaluation five users have been observed. The objectives given to the users were (1) to construct the network composed of two data, two relations, two readers and one writer as shown on the Figure 37 - Network asked to be constructed. (2) Then to edit the properties of data2, relation2 and the writer connecting them, (3) add relation2 to a skill (the skill had to be created first), (4) enable/disable the skill and finally (5) delete data1, relation1 and the writer. The records are available at http://www.info.fundp.ac.be/~fdelang and http://www.info.fundp.ac.be/~ppurnell/FDNet/ in French.

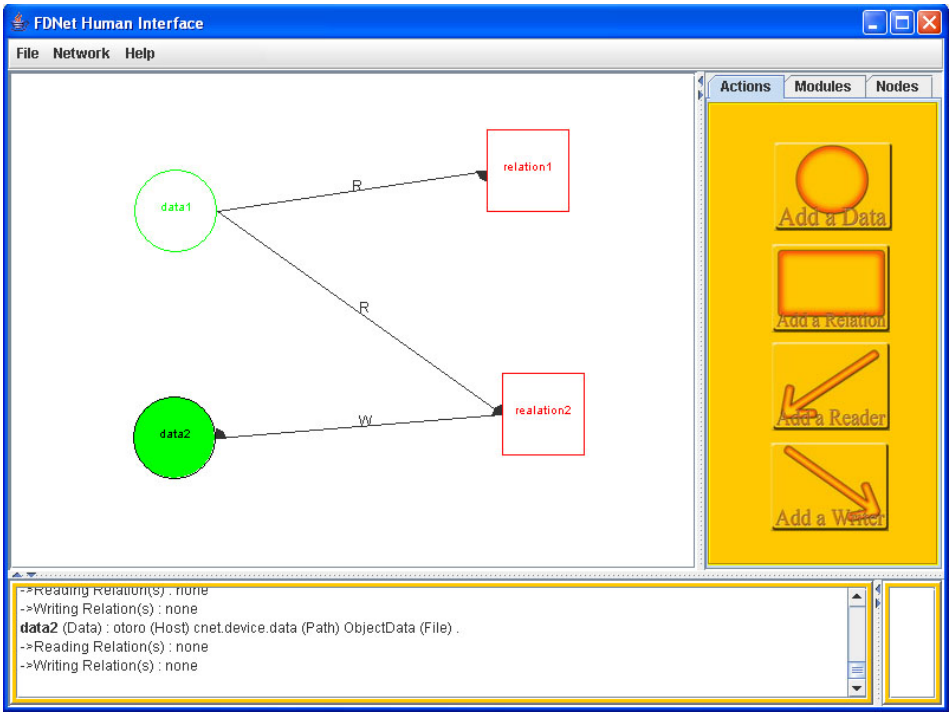


Figure 37 - Network asked to be constructed

(4) Discovered problems

First, almost all the users were directly stopped before the creation of the first node. In fact, the graphical screen shows a blank window just after the HI started. It makes the user think that a new sheet is created to construct the network as other software does. But the user can't create the element and have to go into the Network menu and hit New. By the way, Network is not a good name for the menu in regards of what he contains; users go mostly in the File menu to create a new network.

The difficulties to achieve the delete and the edit property goals (especially for the connections) made almost all the users not able to reach them. Users expected to be able to select the element. Then double click on it and open a property window or use the right click to open a useful menu.

Another problem was shown by a user too accustomed to better HI user when he wanted to validated the new properties. After editing the properties, he hit the return key several times but didn't saw any feedbacks. Actually, this is normal because he had to click on Save with the mouse, but this user underlined a possible improvement to make the HI more alike others existing HI.

In the Network menu the Network Info, Module Info and the Skill Info are not sufficiently explicit. These menus don't make the user think he could find (respectively) the window to add a module, the window to create/delete/edit a Node/Connection and to add a Relation to a skill.

Some users feel uncomfortable because they weren't able to select many elements in the same time and move them.

Finally, the information screen is not displaying the last line of the information. When new lines are written, the scroll bar doesn't follow the last one and stays stick on the first line.

c) Conclusion

From the two evaluations some problems are underlined:

The delete and change property action of the FDNet's HI have to be improved with the use of the right click and an Edit menu in the menu bar.

But that's not all, other methods could be inspired from the others HI's evaluations. For example, there is no keyboard shortcut in FDNet's HI. But the ALT, CTRL and DELETE shortcut are very comfortable for the user.

Plus, there is no zoom in the FDNet's HI. But the zoom functionality provided by HPSim® or Visio® gives a better view of the whole network.

After launching the HI, the HI shows a blank screen. Then when the user wants to create a new network, he simply tries to create an element on the blank sheet. But it's impossible and no feedback appears to ask him to create a new file before.

The menus' names are really not easy to understand and have to be changed to more users friendly.

Moreover, the user cannot select more than one node at the same time and cannot move them freely. He has to move all the nodes one by one.

As said before for the deletion operation, the right click has to be improved and adapted.

Finally, the Help menu does not provide any help.

On the other hand, software like Visio® or Simula® are declined in several version and well working. The evaluation exposed similarities and some disparities theses can be easily completed. This is encouraging for the future of the HI.

I. Improvements of the HI

- To allocate Data and Relation on HI according to which skill they belong to.

  The problem for making an easily reading HI is the number of the components. Some network is composed with more than 250 components. Then component are superposed.

  The idea was to group component by skill, but a problem appears: some nodes may belong to different skills and it doesn't resolve the problem of the superposition.

  Possible improvement is to group components by module (a component can only belong to one module in the same time) and create a new graphical component that can work like a box we open or close (open may be the components visible and close may the components be invisible and regrouped in a same point).

- To display by 3-dimensionals Data, Relation, Reader and Writer which are shown by 2-dimensionals at present.

  We think that if it's very difficult in 2-dimensional to show comprehensively the network, it's also very difficult to do it in a 3-dimensional way. Plus the navigation in the 3-dimensionals environment must be very difficult for novice user.

- Some bugs of the first version remain. For example with a too long XML file some components loose a part of their name.

- Some functionality from the first version seems not to be used. For example the division of component by module.

- Some functionality from the first version is not used. For example the "exist" field of the ConnectionStructure in the EditModule window, the merge method (used to merge the NodeStructures) …

- Provide new functionalities (a zoom function, a print function), improve interactivity (selecting groups of entities, copy/paste), improve the accessibility (interaction with the keyboard), reduce some actions' distance (delete, change properties, change connected Node, create a node, create a network, right click…)

- Review the menus' arrangement and names to make them more understandable (change names…) and user friendly (provide useful help…).

- Create a hierarchy in the frame (make parent-child frame) to make impossible to open several times the same windows. Or, for example, forbid to change information on the graphical screen while the ModuleEdit window is open (this one is constructed with the information we saw on the graphical screen).

- An « Un-do » functionality could be very interesting. We did not integrate it into the present code of the application but a prototype has been attached to the code. It allows to undo/redo the “add a Node” task. We introduce the concept in the appendices: it will give some explanations on how to implement it and show how the function could be integrated to the graphical user interface of FDNet.

IV. Conclusion

The idea of elaborating robots helping rescuers at a disaster's time is a really interesting subject to work on, mostly now that a lot of natural disaster occurs. It gives you a good feeling to being involved in such a great project. We could probably not dream of a better place than Japan to work on robotic systems. Japanese engineers are really skilled people in that specific field and get really involved in their work.

The idea of a pool of different robots (flying, crawling ones) working all together, managed by a single network is really innovative and promising but seems pretty hard to actually realize: it is difficult to adjudicate about FDNet because it did not prove itself as far.

FDNet is a flat distributed network; it could be dangerous because Nodes are not protected against writing. So a crawler robot, developed by someone, could be able to change the properties of some Nodes belonging to a flying robot, developed by another person, and generate some incident. But if the network specifications are strict and if only a few people have access to it, FDNet could be very efficient and could provide new capabilities to engineers because of its original conception. However, if we could attribute locks on the Nodes and institute a hierarchy of the Network, it would be more accessible to engineers, more flexible to different circumstances and more fault-tolerant.

The evaluation showed that the HI is not perfect and gives some difficulties to users (the network is still handmade built, without the HI), but the improvements are not that tough to achieve. In order to help future users and programmers a user's guide with a FAQ (see "Human Interface Manual" pp. 41-62 in the annexes) and a programmer's guide (see "Information to future programmers" pp. 32-40 in the annexes) are provided.

The work done in the last years was built on reliable bases even though it still contains a lot of bugs but this appears because of the long-lasting project. We are really confident in the work that still has to be done to reach the final aim.

Personally, we found this work experience period really enriching because of the cultural adaptations as well as the work surrounding. We found ourselves in real working conditions which allowed us to test what we had learnt in those three past years and it revealed the good level of the formation we had got. To another point of view, the language was a real barrier for the work as well as for our good living but we tried to adapt ourselves the best we could and, according to the feed-back of IRSI, the objectives required have been fulfilled.

V. Bibliography

[1] http://www.rescuesystem.org/tmp/NEW/en/framepage01.htm

[2] Youssef Achbany, Jérôme Jadoulle, “About adding Utility and Usability to FDNet A Flat Distributed Network Architecture”, 2003-2004

[3] http://www.kobe-u.ac.jp/research/reports/sinsai10/vol-5/booth-rescue-robot.pdf

[4] www.extremeprogramming.org/

[Berry and Broadbent 1990] Berry, D.C., and Broadbent, D.E. (1990) "The role of instruction and verbalisation in improving performance on complex search tasks", Behaviour & Information Technology 9, 3(May-June), pp. 175-190

[CLARAty] Tara Estlin, Rich Volpe, Issa Nesnas, Darren Mutz, Forest Fisher, Barbara Engelhardt, and Steve Chien, “Decision-Making in a Robotic Architecture for Autonomy” Jet Propulsion Laboratory available at http://robotics.jpl.nasa.gov/tasks/claraty/overview/publications/01_estlin_decision_isairas.pdf

[EXCENTRIC LABEL] Fekete, J.-D. and Plaisant, C. (1998), *Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualization*, Proceedings of CHI'99, Pittsburgh, PA, USA, May 15-20, 1999, ACM, New York, 512-519.
HCIL-98-09, CS-TR-3946 , UMIACS-TR-98-59

[FBODART] http://www.hec.unil.ch/fbodart/IHM/Chap3/crit.html

[Fujia & Kageyama 1997] Masahiro Fujia and Koji Kageyama, "An Open Architecture for Robot Entertainment", *Proceedings of the First International Conference on Autonomous Agents*, ACM Press, 1997

[HPSIM] HPSim Version: 1.1 Copyright (C) 1999 - 2001 Henryk Anschuetz

[HPSim eval] Copyright© HPSim-developers 2002 available at http://www.adobe.de/products/acrobat/readstep2.html

[Inukai 2003] Toshihiro Inukai, “ORiN: A common object model for robotic systems”, http://staff.aist.go.jp/t.kotoku/fyi/AIM2003.html#TANIE (20 July 2003) (Date of access May 2004)

[JARA 1999] JARA, “Specification of Orin”, http://www.jara.jp/E_ORiN/En_ORiN.htm (1999)(Date of access May 2004)

[Lambot] Lambot Nicolas, “FDNet: Enhancing Human Interface with Dynamic Capabilities” 2003

[LEWIS97] Clayton Lewis, John Rieman 1993. *"Task-centered user interface design, a practical introduction"*. Disponible à ftp://ftp.cs.colorado.edu/pub/cs/distribs/clewis/HCI-Design-Book/.

[NIELSEN 93] Jacob Nielsen, *" Chapter 5: Usability Heuristics "*, in Usability Engineering, Academic Press, 1993

[NORMAN 86] D.A. Norman, *"Cognitive Engineering "*, in [NORMAN-DRAPER 86].

[NORMAN-DRAPER 86] D.A Norman., S.W. Draper , *" User Centered System Design on Human Factors Interaction : New Perspectives "*, Lawrence Erlbaum Associates Publishers, Hillsdade, 1986.

[SIMULA] SIMULA version1.0 Copyright (C) 1999 – 2000 Design en Programmig, Eng. Mohamed Ali Sobh, available on http://mathtools.emediaworld.net

[Ozaki 2003] Fumio Ozaki, “Open Robot Controller Architecture (ORCA)”, http://staff.aist.go.jp/t.kotoku/fyi/AIM2003.html#TANIE (20 July 2003) (Date of access May 2004)

[Toshiba Corporation 2003] Toshiba Corporation, “Toshiba to Bring its New “ApriAlpha” Concept Model Robot to ROBODEX 2003”, http://www.toshiba.co.jp/about/press/2003_03/pr2001.htm (20 March 2003)(Date of access May 2004)