Info-bead User Modeling

By: Eyal Dim and Tsvi Kuflik

Table of Contents

1 INTRODUCTION

The idea of user modeling servers was proposed to allow sharing of user modeling functions and content. While user modeling research led to important contributions, current systems that provide personalized services to their users still rely on their own proprietary UMs developed in an ad-hoc manner. The main reasons for this are: (i) System developers focus on the specific characteristics of users in order to deliver a specific service (e.g., a movie recommendation); (ii) reusable user modeling components are not publicly available; (iii) the user modeling community does not have commonly accepted standards for UMs and their structure; and finally, (iv) the development of a standard and reusable UM that holds a comprehensive set of information-items requires a level of effort that is too high to be carried out by a single application.

To allow the development of UMs from reusable components, we propose to adopt a component-based software development approach. In particular, we propose developing and maintaining small, standard and reusable multipurpose building blocks that may be integrated into more abstract UMs as needed. We use the metaphor of information pendants (info-pendants) made of information beads (info-beads) and their connecting threads (info-links). More specifically, an info-bead is a standalone and encapsulated module that uses inputs or default data to infer a new piece of information about a user, and possibly delivers this information to service applications or to other info-beads linked to it. Moreover, an info-bead may use any inference mechanism to generate its output information. An info-pendant is a composition of info-beads, where data flow from one or more info-beads to another info-bead, invoking the latter's inference process and information generation. Consequently, the info-pendant becomes a net of user-related info-beads. For example, an info-pendant for location-based services may consist of four info-beads: GPS-position-coordinates, street address, mobile phone cell, and tagged location (e.g., "at home," "at work," "at the mall"). The information is provided by the four specific info-beads to another info-bead that reasons on their outputs and provides the best information available as needed (for instance, a tagged location may be used if GPS data are not available).

The availability of a collection of info-beads and info-pendants may encourage the UM providers to adopt standard info-beads for threading a new UM. Furthermore, group models (GMs) may thus become a natural extension of individual UMs. Reusability, in this case, refers not only to being able to associate the model with different users or to share the same information in multiple contexts, but also to being able to construct a variety of partial UMs from a collection of info-beads and info-pendants suitable for specific services. Constructing UMs from simple and small info-beads may allow the recruiting of a multinational effort to map and generate them, offering an opportunity for better collaboration among UMs.

2 Introduction to Info-bead User Modeling

This introduction to componentization of user models starts by defining the basic elements in the info-bead user modeling approach, followed by a presentation of the internal structure of info-beads.

2.1 Basic Elements

The info-bead user modeling approach refers to five main elements: info-beads, info-links, info pendants, info-bead UMs, and info-bead GMs. This section starts with a definition of those elements, while their implementation details are presented in the following sub-sections.

> *Definition 1:* An **info-bead** is an atomic user modeling component that supplies a single attribute of a user (or a group of users).

For example, an info-bead may return the user's blood pressure, another info-bead may return her level of happiness, and two other info-beads may refer to whether the user is involved in a leisure activity and to her current location.

> *Definition 2:* An **info-link** is a data exchange protocol that allows the transmission of an attribute's data from one info-bead to another.

For example, an info-link may allow a location info-bead to send the location data to the leisure activity info-bead, which infers the probability that a user is involved in the leisure activity.

> *Definition 3:* An **info-pendant** is a tree of info-beads, linked by info-links that support the generation of the attribute handled by the root info-bead.

For example, an info-pendant may infer the user location from info-beads that handle GPS, Wi-Fi-based, and cellular positions. In this case the info-beads that handle the GPS, Wi-Fi-based, and cellular positions will be in the leaves of the info-pendant's tree, and the user location info-bead will be the root of the info-pendant. Info-links will connect the leaves to the root.

> *Definition 4:* An **info-bead UM** is a collection of info-beads partially linked by info-links to generate networks that hold attributes of a single user.

For example, an info-bead UM for a medical domain may contain a patient's blood pressure info-bead and, in addition, the location info-pendant for locating the patient within a hospital.

> *Definition 5:* An **info-bead GM** is a collection of info-beads partially linked by info-links to generate networks that hold group attributes, info-bead UMs representing members in the groups, and info-bead GMs representing sub-groups.

For example, an info-bead GM of a students' class may contain an info-bead GM of students (that contains info-bead UMs, each representing a different student), an info-bead UM of the teacher, and an info-bead of the average grade of the class.

To allow fluent reading, the acronym UM is used for both info-bead UM and other UMs, and the term GM for both info-bead GMs and other GMs.

2.2 The Internal Structure of Info-beads

An info-bead is composed of three parts: the **operational part**, the **metadata part** and the **control part** (see Figure 1).

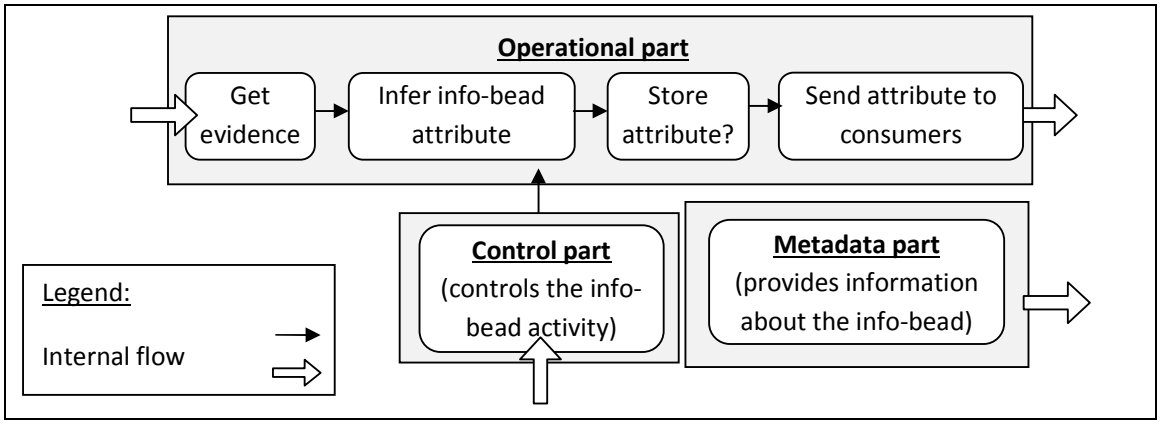

Fig. 1. The internal structure of an info-bead

The operational part of an info-bead receives **evidence data** through **input interfaces** that provide information from other info-beads, or it uses internally an API (Application Programming Interface) of external sources (e.g., sensors or HMI – Human Machine Interface) to obtain the evidence data. The info-bead turns the data into a **value** of an attribute through an **inference** (or calculation) process. There are no restrictions on the inference process, and it is left to the info-bead developer to decide how to implement it. The info-bead outputs **triplets** of {ID, detection-time, information-item}, where **ID** identifies the

source of information and **detection-time** is the date and time when the triplet is set. The **information-item** includes the evidence data, the value (information) generated by the info-bead, data on the value type (which can be atomic or composite, e.g., temperature, level of excitement, or x, y, z coordinates of a GPS position), its units, its accuracy, and a confidence level (in case of uncertainty with regard to the data attribute). The information-item further stores time properties, such as expiration time and time resolution, which are needed to understand better the accuracy and the resolution of the time itself. An info-bead may store the triplets in short-term or long-term storage, depending on the info-bead implementation. When needed, a triplet can be sent to other info-beads through the **output interface** of the info-bead, or to **an external consumer** (i.e., a personalized application).

The **info-bead control part** may have the following functions: (1) Activation/ deactivation of the info-bead (which respectively changes the status of the info-bead to on/off); (2) authorization to access its information (according to privacy and security considerations); (3) definition of its communication method with other info-beads (i.e., push, pull, or notify connection types); (4) initialization or updating of the info-bead's settings (e.g., identifying a Bluetooth address); (5) allowing authorized applications to access the explanation of the info-bead's information based on the stored evidence; and (6) maintenance operations of adding or deleting information.

Finally, the **info-bead metadata part** includes information on the info-bead, such as its name, its keywords, its value units, a link to a help file that explains the info-bead functionality, inference process and interfaces, the names of its input interfaces, and the name of its output interface. The metadata also include the provider information, such as the part-number (an identification given to the info-bead by its provider), its version, its backward compatibility with previous versions, the web resource (URL) where the info-bead may be found, the provider's contact information, and the authentication trustworthiness data (proof that the info-bead is authenticated and can be trusted). All this information is required for retrieval purposes. Furthermore, this information is needed for the correct connection of reusable info-beads, which are provided by multiple providers, into a UM or GM.

As an example of how an info-bead works, let us consider an info-bead that contains information about the user's age. The metadata of the "Age info-bead" would contain stable information (such as, info-bead's name: Age; keywords: age, birth date; help file: a link to the help file explaining its inference process; info-bead's version: v1.0). These would help a designer to search for info-beads relevant to a specific user-modeling task. If the info-bead's control part is set to be active, and if authentication and security requirements permit, the operational part would get two evidence sources: the user's birth date (from another info-

bead) and the current date (from the calendar API). The inference process would subtract the birth date from the current date, extract the number of years and result with the age value. The info-bead would add the required data into the information item in the form of an age value, including the evidence data (birth date and current date), value type (integer), value units (years), value accuracy (1 year), and the confidence level (1.0 for full confidence). Furthermore, time properties would be added: the expiration time (next birthday), inference time (current time), time-representation resolution (1 millisecond), and more. Depending on the decision of the info-bead developer, the age value would (or would not) be kept in the info-bead's storage. Then, if the info-beads control permits (for privacy and security considerations), the age attribute would be forwarded to other info-beads within a triplet containing: {the sending info-bead ID, the detection time (current time), information-item containing the age value}. When the mode is push, for example, the info-bead inference process may sleep until the next birthday or until there is an update of the birth date, and wake up to send a new update of the age attribute.

From an implementation point of view, info-beads can be implemented as Java classes that extend an abstract info-bead class. Triplets may be sent to other info-beads or service applications using the selected communication method (e.g., using Java Observer design pattern for the "push" communication method).

3 The Info-bead User Modeling Approach

The info-bead user modeling approach presents the composition of info-beads into info-pendants, UMs, and GMs. Next, it elaborates on the association of UMs and GMs with specific users or groups.

3.1 Connecting Info-Beads to Info-Pendants, UMs and GMs

Info-beads are connected using info-links into info-pendants. An **info-link** connects exactly two info-beads, the first of which (the sending info-bead) is called "**source**" and the second (the receiving info-bead) "**target.**" An info-link may be **valid** or **invalid**. An info-link is **valid** if and only if the **output interface** of the source info-bead fits one of the **input interfaces** of the target info-bead. The implementation of an info-link depends on the communication connection type (for example: push, pull, or notify). There is a **path** between two info-beads if the output data of one triggers eventually (possibly passing through other info-beads and info-links) a change in the other's input data. The root info-bead in an info-pendant, namely, the info-bead whose output is the expected user's attribute value, is called **attribute holder** and identifies the entire info-pendant.

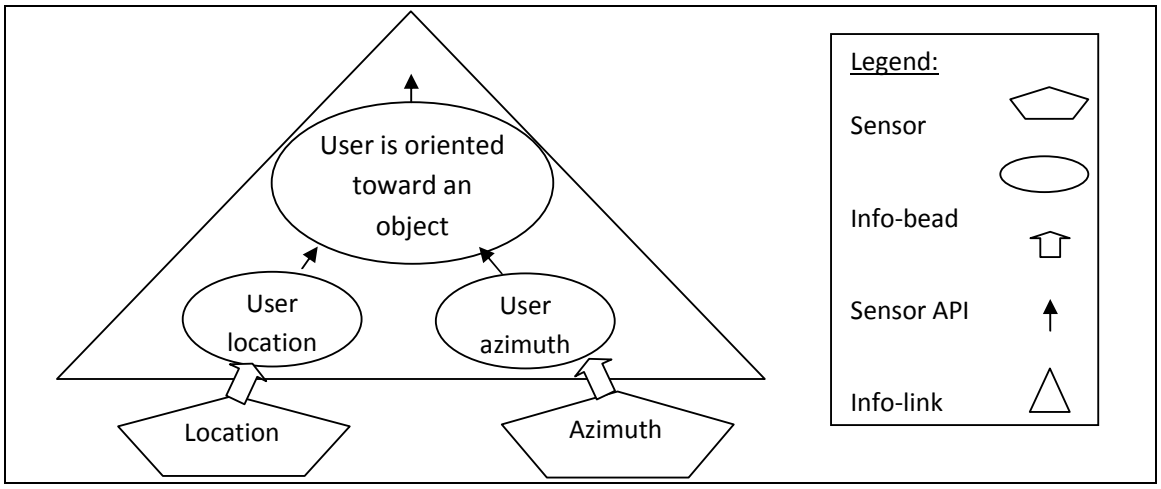

Fig. 2. A simplified example of an info-pendant

An info-pendant may be encapsulated, revealing only the values of its attribute holder, or it may be revealed, so that each of the constituent info-beads can provide its (output) information too. The info-pendant does not have a functional task when the model is activated. It is only a virtual container, which facilitates the UM designer's work, meant to allow a collection of info-beads to be viewed, saving or retrieving and info-links that keep the info-pendant characteristics.

As an example of an **info-pendant**, let us assume that the user is located in a space populated with objects. The mapping of the objects in the space is known in advance, and the required user attribute is whether the user is oriented toward an object (which may be

important in an exhibition or shopping scenario). Figure 2 illustrates an info-pendant that infers whether the user is oriented toward an object. This info-pendant is composed of three info-beads. The sensors shown at the bottom of Figure 2 detect the location and azimuth of the user. The "user location" info-bead is directly connected to the sensor through its API. It gets the "location" sensor data through an API and identifies user location. Similarly, the "user azimuth" info-bead gets the "azimuth" sensor data through its API and identifies the user's azimuth. A third info-bead, shown at the top of Figure 2, which is the attribute holder of the info-pendant, outputs the requested user's attribute, namely whether the user is oriented toward an object. To this end, this info-bead acquires the location data from the "user location" info-bead and the azimuth data from the "user azimuth" info-bead. It compares the user location with the mapping of objects' positions, and if the user is in the proximity of an object, it compares the user azimuth with the heading from the user location to the object's position and infers whether the user is oriented toward the object. It should be noted that an info-pendant has a single info-bead, the attribute holder, that is not a source of any info-link in the info-pendant. Any other info-bead within the info-pendant must be a source and must have a target in a path of info-links leading to the info-pendant's attribute holder.

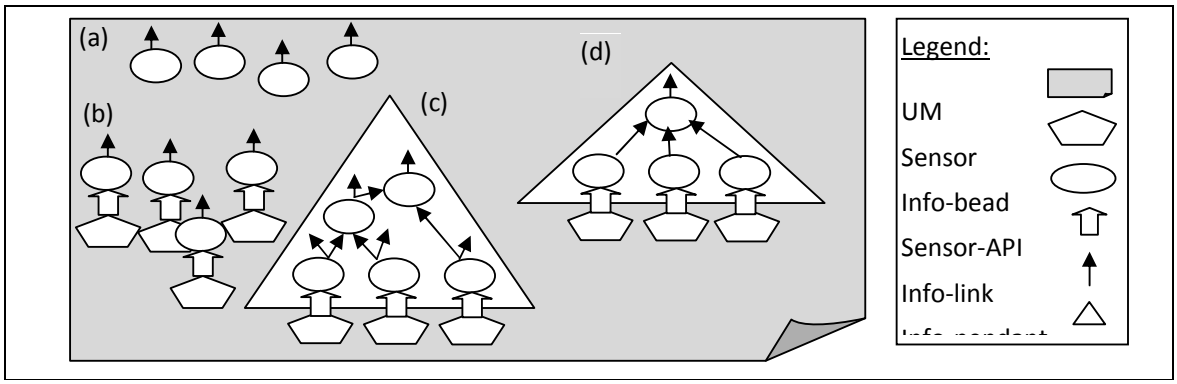

Fig. 3. A schematic example of a UM

Figure 3 shows an example of a UM. Figure 3(a) shows info-beads that do not have any input, providing constant, predefined data (e.g., that the user is a human). Figure 3(b) shows info-beads that get sensor data through the sensor API. Figure 3(c) demonstrates an info-pendant that reveals its info-beads' information (the top vertical arrow on each info-bead), and the info-pendant in Figure 3(d) reveals only its attribute holder value.

As noted, a **GM**, which represents a group of users, includes the UMs of the group members or the GMs of sub-groups. In addition, it may include info-beads or info-pendants representing the group attributes. These are attributes that are relevant to the whole group and not to the individual members, such as group type (e.g., family members, work colleagues,

and social friends), or cohesiveness (how tight the relationships among the group members are). Figure 4 presents a GM (in dark gray) that contains several UMs (bottom part, light gray). In addition, it contains one info-pendant and three info-beads (top part, white) that handle group attributes. Info-beads within the UMs may be linked, if needed, to group info-beads delivering information required for the inference done within the group info-beads, as illustrated in Figure 4.

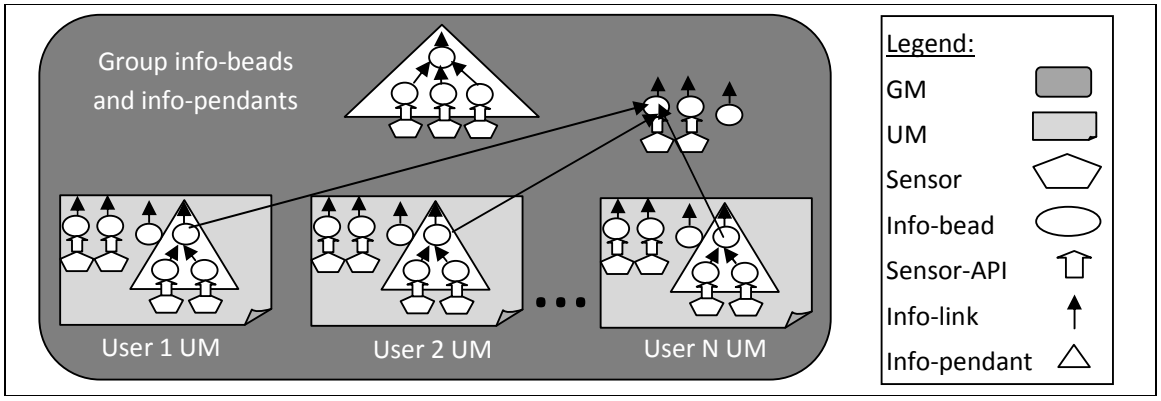

Fig. 4. A schematic example of a GM

3.2 Association of UMs and GMs to Individuals and Groups

A UM may be embedded in an application, collecting data about its users. This may be done when an application uses a UM and associates it with an individual user. Once the UM is activated, the "user ID" info-bead is instantiated. Besides the specific **user** to which the UM is associated, the UM belongs to an **owner**, namely, a person, an organization, or an application that is interested in the data and responsible for its control and maintenance. The owner may be the user herself or any other entity. Furthermore, a UM may have more than one owner, as when it has a user and an administrator who can update the specific UM information. The owner is detected by adding an "owner ID list" info-bead.

When associating a UM to a user, its sensors may need to be initialized to the specific person as well (e.g., setting the phone number). The connection between the info-bead and the sensor is done through the API of the sensor.

A GM is associated with a specific group in a similar way. The GM represents the specific group and can be owned by several owners, e.g., when each group member is authorized to control the GM's info-beads data, or when the GM belongs to an organization, in which case the organization's representatives or the service application may become the owner. A group info-bead that is associated with a sensor should initialize the sensor ID as well.

As an example of the association of a GM, consider a basketball team of five people. The GM may contain the team's game-score. This GM would need to identify five specific players, associate them with their UMs, and a specific sensor that collects their score into a group info-bead. Since two competing groups of five players participate in a basketball game, there should be two associations of the GM, one for the first group and one for the other group. From the point of view of game management, a referee (a real person) should be the owner of all UMs and GMs.

4 The Generation of An Info-bead User Model

The info-bead user modeling approach involves three types of roles, who perform different activities: (1) info-bead developers, (2) UM designers, and (3) application developers.

An **info-bead developer** is usually a programmer who develops info-beads. The developer is free to use any programming language and should be able to build any inference mechanism to be included in an info-bead. The developer is expected to develop a new info-bead if such an info-bead is not found in info-beads repositories. A search of such repositories may be conducted using the info-beads' metadata. The metadata may also serve other developers by finding "close enough" code examples, decreasing rework.

A **UM designer** is responsible for designing info-pendants, UMs, and GMs by connecting and linking info-beads. A UM designer needs a UM design tool, such as that presented below, as well as knowledge about the specific domain and the info-bead user modeling approach. The design tool would allow the designer to assemble a UM or GM without having software programming knowledge. The UM designer searches for ready-made info-beads or info-pendants using the supplied metadata or help files (similar to the way a developer searches for them), and links them into new info-pendants that may then be saved for future use. If a relevant info-bead does not exist, the UM designer should request an info-bead developer to develop it. The UM designer can further aggregate required readymade info-beads and info-pendants and create UMs and/or GMs, which may also be saved for future use.

An **application developer** is a programmer who uses readymade UMs or GMs relevant to a given application, integrates them into the application's functionality as "black boxes," and associates them to specific users or groups.

Although all three roles are important for creating personalized applications, the info-bead user modeling approach shifts the focus from the application developer to the info-bead developer and the UM designer. The latter, who does not need to have software development skills, requires special support to carry out the entailed activities. Thus, we have developed a supporting **UM design tool**: the "Info-bead User Model Editor." A user manual of the tool is available in Section 5, and a short description is given here.

5 The Info-bead UM User Manual

5.1 Introduction

The Info-bead UM Editor (IBUMCORE project) allows a UM designer to design a user model (UM) or a group model (GM) from readymade info-beads. It allows to save info-pendants, user models and group models. It also allows to automatically prepare a user or group model ready for activation.

5.2 Installation

The Eclipse project for the editor is called IBUMCORE. It was developed using Eclipse Java EE IDE for Web Developers Version: Juno Service Release 2. It uses JAVA SE-1.7. The Info-bead UM Editor prototype was implemented using JUNG 2.0.1[1] for displaying the graph of the user model or the group model. To work with the editor please make sure that you have downloaded and installed Eclipse from: https://www.eclipse.org/downloads/ .

The IBUMCORE JAR for Eclipse may be downloaded from XXXXXXXXXXXXXXXX . Once downloaded the class DirectoriesSetup in package architectureUtil should be changed to fit the new location of the IBUMCORE in the target computer. The parameters that should change are: ECLIPSEWORKINGDIRPATH and ECLIPSEWORKINGDIRPATHURL.

5.3 The "elements" directory

Table 1 : List of sub directories in the "elements" directory

Directory	Purpose	File type
activatedModels	Holds an XML file of the model saved for activation.	eam
groupModels	Holds XML files of saved group models	egm
Help	Holds text files for the help function of the editor	txt
infobeadMetadata	Holds XML files that contain the names of the input links and the output links of each info-bead	metadata

[1] https://code.google.com/p/jung/wiki/Manual accessed December 30th, 2013

infoPendants	Holds XML files of saved info-pendants	eip
otherFiles	Allows to access any other file if needed	Any type
userModels	Holds XML files of saved user models	eum

The IBUMCORE project contains a directory by the name "elements". This directory holds the sub-directories presented in table 1.

5.4 Running the applications

There are two applications to run: the EditorMain application and the ActivatedModel**1**Main application. The EditorMain application runs the info-bead graphical editor itself. The AcivatedModel1Main application runs the currently saved model for activation. Each run would generate a new independent instance of the user model or the group model.

5.5 An overview of the editor screen

The editor enables the user to access readymade info-beads (see Figure 7a). Once info-beads are available, the user may select and drop them into the working area (Figure 7b). Info-beads that acquire their input data only from sensor API are represented in green (Figure 7c). It should be noted that the info-beads encapsulate their connection to the sensor, and therefore the sensors are not shown in the info-bead editor. Info-beads that have standard input link/s and forward data through an output link are presented in blue (not shown here). The editor **automatically** generates **valid** info-links among the selected info-beads, based on the input and output interfaces of the info-beads. Figure 7d presents the valid links identified by the editor. The valid links are displayed as gray, dashed arrows. The user is required to select which info-links to use (activate). Selected info-links are presented as solid arrows (see Figure 7e). If the result is a single info-pendant (i.e., all info-beads in the working area are connected in a tree-like format yielding a single output), the user may save the info-pendant for future use. Similarly, a collection of info-beads, info-pendants, and UMs (representing different aspects of the same user) in the working area may be saved as a UM, and a collection of info-beads, info-pendants, UMs, and (sub-)GMs may be saved as a GM. The info-beads, info-pendants, UMs, and GMs may be loaded to the working area by selecting them from the relevant tab (Figure 7f). The editor allows sub-models to be selected and dropped into a GM.

The editor has also standard menus (Figure 7g), including the file menu, the edit menu, the view menu, the run menu, and the help. An information pane (Figure 7h) presents the UM designer's selected actions. Finally, a status bar presents information about an info-bead that is selected by the user (Figure 7i).

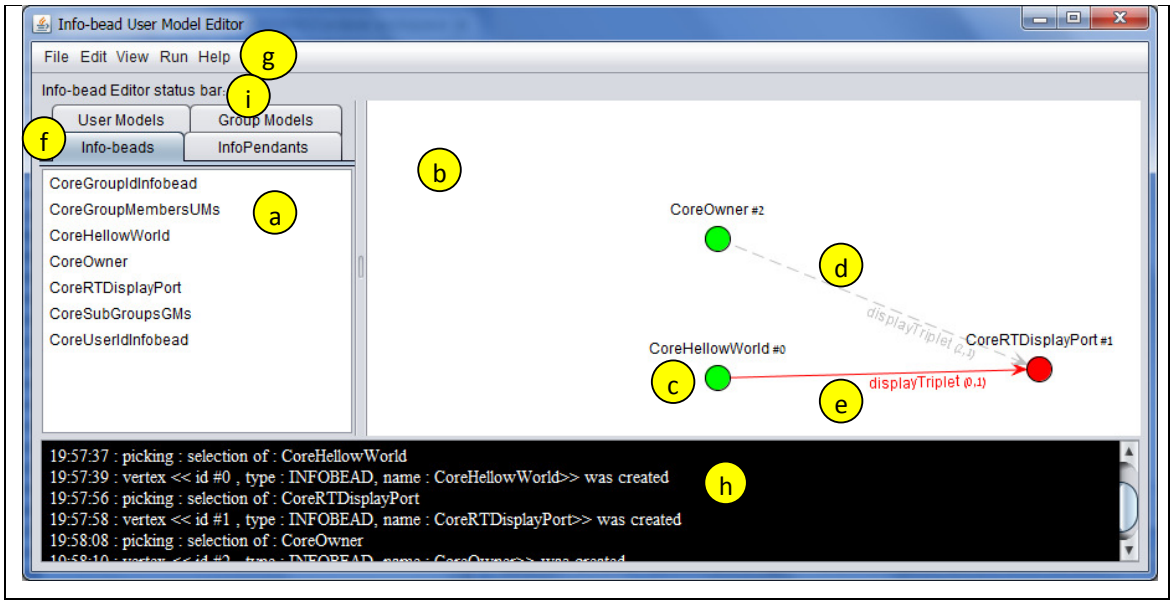

Fig. 5. The Info-beads UM editor

5.6 Developing an info-bead

Developed info-beads are stored in the IBUMCORE/src/infobeadCollection package. If developed elsewhere, they should be imported. The info-bead is identified by its class name. Two additional files must be prepared: a metadata file, describing the input info-links and the output info-links, and the help file, describing the behavior of the info-bead. The metadata file must be kept in the elements/metadata directory described in table 1 above, and the help file in the elements/help directory described above. The editor uses the metadata file to decide what the legitimate info-links are, and the help file to present the help menu.

Hello world info-bead example: Let us develop the **CoreHelloWorld** info-bead. This is a simplified info-bead. It is presented in Figure 6. This example info-bead counts up to 15, and on each count sends the counter through the pushData method. As mentioned above, the developer must add a CoreHelloWorld.metadata file with the input info-links and output info-links (in this case, we have only an output info-link). In addition, the developer must add a CoreHelloWorld.txt file with the help information. Both files should be added in the correct directories as presented in Table 1 above.

```
package infobeadCollection;
```

```java
import java.sql.Time;

import genericInfoBead.InfoItem;
import genericInfoBead.Triplet;
import genericInfoBead.InfoBead;


/**
 * Example of how a sensor info-bead sends data to a display port connected to it
 * @author Eyal
 * @vesion  1.0;
 *
 */
public class CoreHellowWorld extends InfoBead implements Runnable {

	private static final long serialVersionUID = 1L;


	public void initialize() {
		Thread helloThread = new Thread(this,"HelloWorld");
		helloThread.start();

	}

	public void handleData(Triplet tripletTest) {

	// no need for a sensor

	}


	@Override
	public void run() {
		Triplet tripletTest = new Triplet("Hello World Info-bead");
		for (int n=1;  n < 15; n++){
			int randomSleep = 500 + (int)(Math.random() * ((2000 - 500) +
				1));
			System.out.println("<HelloWorld> value is : " + n + " Sleep: "
				+ randomSleep + "\n");
			Time t = new Time(System.currentTimeMillis());
			InfoItem data = new InfoItem();
			data.setInferenceTime(t);
			data.setExplainInfo("Main >> Testing Display Services");
			data.setInfoType("test");
			data.setInfoUnits("testUnits");
			data.setInfoValue(n);
			tripletTest.setTime(t);
			tripletTest.setInfoItem(data);
			pushData(tripletTest);

			try {
				Thread.sleep(randomSleep);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}
}
```

Fig. 6. The hello world info-bead

It is a good practice to name the info-link by the same name as the sending info-bead. However, there is one exception, the **CoreRTDisplayPort** info-bead. This info-bead (appears in Figure 5) is used as a "standard output" info-bead for testing purposes. Therefore, its input info-link may appear as an output info-link of every info-bead.

An example of a metadata file for the CoreSubGroupGMs info-bead is given in Figure 7. This info-bead has a single input info-link and two output info-links, the one that delivers the info-beads value to other info-beads, and the one that delivers the value to the **CoreRTDisplayPort** info-bead.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<metadata>
    <output-interface>displayTriplet</output-interface>
    <output-interface>CoreSubGroupsGMs</output-interface>
    <input-interface>CoreGroupIdInfobead</input-interface>
</metadata>
```

Fig. 7. A metadata file example for the CoreSubGroupGMs info-bead

Common Design Errors:

- Forgetting to add the metadata file to the elements/metadata directory would cause the info-bead to disappear from the info-beads list when the info-beads tab is selected (Figure 5f).
- Adding a metadata file with a name different from the info-bead name would result in a "class not found" exception when running a user-model containing the info-bead.
- Inaccurate name of an input or output info-link would result in a missing info-link edge in the model graph (connecting the sending info-bead with the receiving info-bead).
- Missing help file would result in an error message when help is selected through the help menu.
- The character "_" is reserved and should not appear in the info-bead's name.

The hello world example above presents a simplified info-bead structure. To assist in development, an info-bead template is given in the **CoreTemplateExample** info-bead class.

5.7 The editor

When the EditorMain class is selected for run in the Eclipse, the editor runs and presents the message in Figure 8, to remind the user to start a new user or group model or to open an existing model that was previously saved.

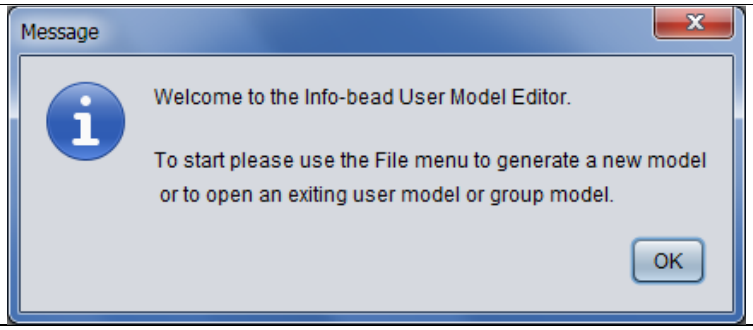

Fig. 8. The message displayed when the editor is first started

5.7.1 The File menu

The user selects the option from the **File menu**, presented in Figure 9. It has the following options:

- New user model – Opens a new user model in the working area
- New group model – Opens a new group model in the working area
- Open user model – Opens a file selection menu to select a user model from the directory of previously saved user models
- Open group model – Opens a file selection menu to select a group model from the directory of previously selected group models
- Close – Closes the model in the working area
- Save – saves the current model under the same name
- Save model as – Saves the current model under a new name. If it is a user model, it will be saved in the elements/userModels directory by default. If it is a group model, it will be saved in the elements/groupModels directory by default.
- Save info-pendant as – Saves the info-pendant containing the selected info-bead in the elements/infoPendants directory.
- Delete model or info-pendant from list – Deletes a selected info-pendant from the info-pendants list if the info-pendants tab is selected (Figure 5f). Deletes a selected user model from the user-models list if the user-models tab is selected. Deletes a selected group model from the group-models list if the group-models tab is selected.
- Exit – Exiting the editor application.

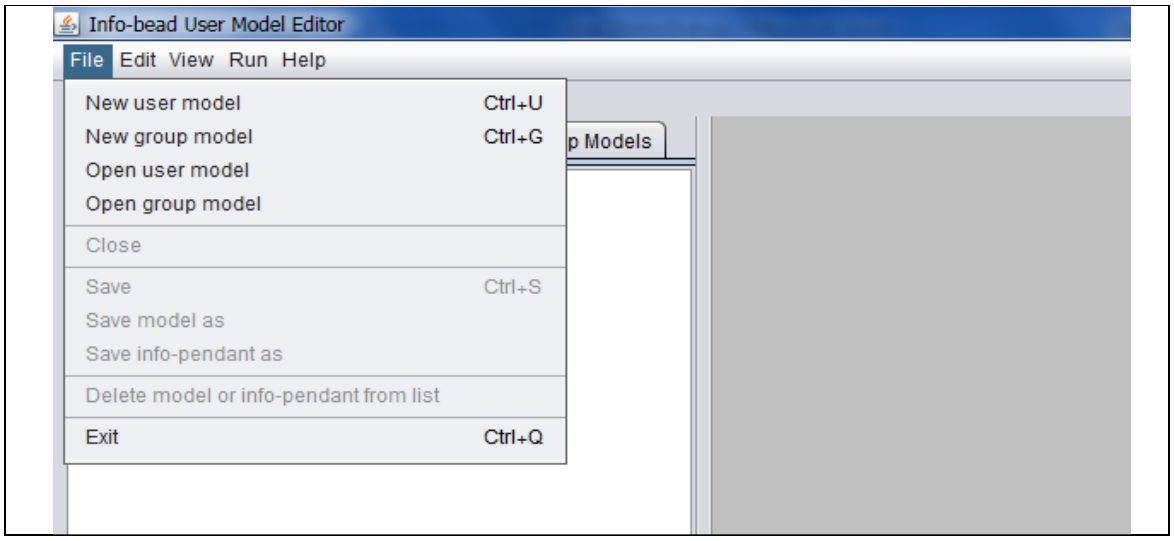

Fig. 9. The File menu

5.7.2 The selection list tabs

There are four **tabs** (Figure 5f) in the editor. Selecting a tab opens a list of items (Figure 5a). This are the four lists:

- Info-beads – A list of info-beads in the application. The info-beads names are taken from the elements/metadata file-names. A selected info-bead may be dropped in the working area. Once dropped there, the possible info-links will appear as a dashed arrow (Figure 5d). The user should select the required info-links turning them into solid arrows to allow them to be active. In a user model activated for a specific user (or a group model activated for a specific group), data would flow only through active links. The info-bead that has been dropped in the working area would belong to the current model. The same info-bead may appear more than once in a user model or a group model, therefore the info-bead's name is concatenated with a unique ID (unique within the model) preceded by the character "#", for example: the info-bead CoreOwner appears as CoreOwner#2 in Figure 5, identifying the ID as 2.
- Info-pendants – A list of info- pendants previously saved through the File menu in the elements/ infoPendants directory. A selected info-pendant may be dropped in the working area. In such case, all its info-beads would get a new id and would belong to the currently edited user model or group model.
- User models – A list of user models that have been previously saved in the elements/userModels directory through the File menu. A user model may be selected and dropped in the working area. If the user is currently editing a user

model in the working area the content of the user model would be copied into the working area, and its info-beads would get new id. If the user is currently editing a group model in the working area, the user model content would be copied into the group model, but they will still belong to the user model. The user model would become a sub-model of the group model. The IBUME is set so that it would not offer connections among user models which are sub models of the entire group model.

- Group models - A list of group models that have been previously saved in the elements/groupModles directory through the File menu. A group model may be selected and dropped in the working area only if the user is currently editing a group model. In this case, the group model content would be copied into the working area and given a new id. However, the info-beads of this group model would still belong to it, and the dropped group model would become a sub-model of the currently edited model in the working area.

5.7.3 Editing a user model or a group model and its impact on the status bar

Figure 10 presents a user model that contains a single info-bead, the CoreUserIdIndobead. The status bar appears above the tabs and below the menu bar (rounded by the red ellipse in Figure 10).

For the user model, the status bar presents the following information about a selected info-bead (graph vertex) within the working area (the selected vertex color turns to yellow):

- The type of model that is currently being edited in the working area (a user model).
- The selected vertex name, which is the info-bead's name
- The selected vertex ID, which is the info-bead's ID

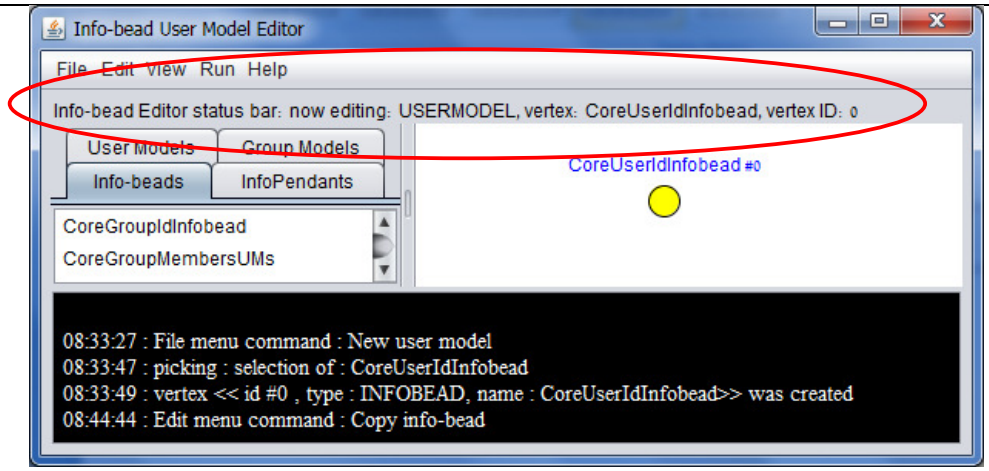

Fig. 10. The user model status bar

Figure 11 presents the case of a group model editing. The first step of the group model editing, presented in Figure 11A, is to add a user model as a sub model. The user model "u1" is selected from the user models list (User Models tab), and dropped into the working area. It contains two info beads, the CoreHelloWorld info-bead and the CoreRTDisplayPort info-bead. When the user selects the vertex in the graph which represents the CoreRTDisplayPort info-bead, the status bar shows that: a group model is currently being edited, the selected info-bead is CoreRTDisplayPort, and that its ID is 7. Furthermore, the status bar shows that the vertex belongs to a user model (which is a sub model of the currently edited group model), that the sub model's ID is 13, and that the ID of the parent model (which is currently edited group model) is 10. In this case, the user decides to add another "u1" user model to the group model. The resulting group model (Figure 11B) includes sub models of two users, i.e., a group of two. When the vertex representing the CoreRTDisplayPort of the second user model is selected, the status bar shows that its ID is different than the ID of the same info-bead in the first user model, and that its model ID is also different (=16), while the parent group model ID is still the same (=10). Finally, the user of the editor adds the CoreRTDisplayPort info-bead as a group info-bead (Figure 11C) from the info-beads' list (using the Info-beads tab), and when the vertex of this info-bead is selected, its model ID is 10, the currently edited group model, and its parent model is null, as it is the root model.

Fig. 11. The group model status bar

To summarize, for a group model, the status bar presents the following information about a selected info-bead (graph vertex) within the working area (the selected vertex color turns to yellow):

- The type of model that is currently being edited in the working area (a group model).
- The selected vertex name, which is the info-bead's name
- The selected vertex ID, which is the info-bead's ID
- The selected info-bead model ID
- The selected info-bead parent model ID, if the info-bead is part of a sub-model within the currently edited model. Otherwise, null if there is no parent model.

Common Design Errors:

- Adding an info-bead to a user model that is a sub-model of the group model while editing the group model, would result in that info-bead being part of the group model and not of the user model. The right way to do this is to edit the user model add the info-bead to it, and only then to select and drop the user model within the group model.

5.7.4 The Edit menu

The Edit menu is presented in Figure 12. It has the following menu items:

- Reset info-bead colors – Allows to return the info-beads to their original color (see next menu item to see when the color may change).
- Find info-bead – Allows to enter any string (case sensitive). Next, the color of any vertex representing an info-bead would turn orange if the string appears within the info-bead's name, making it easier to find it in the graph.
- Show info-pendant - Selection of this menu item allows showing the info-pendant of the currently selected info-bead, if such info-pendant exists.
- Show sub-model –Selection of this menu item allows showing the sub-model of the currently selected info-bead, if the user is currently editing a group model.
- Copy info-bead – Allows copying the currently selected info-bead.
- Paste info-bead – Creates a new vertex for the copied info-bead. The new-pasted info-bead gets a new ID. It is also created in the currently edited model, so it gets the edited model ID (that may be different from the model ID of the original copied info-bead. The editor also suggests the possible info-links.
- Delete info-bead – Deletes an info-bead and all its info-links from the graph.

- Delete sub-model from group model – If the currently edited model is a group model selection of this menu item would delete the entire sub model of the currently selected info-bead.

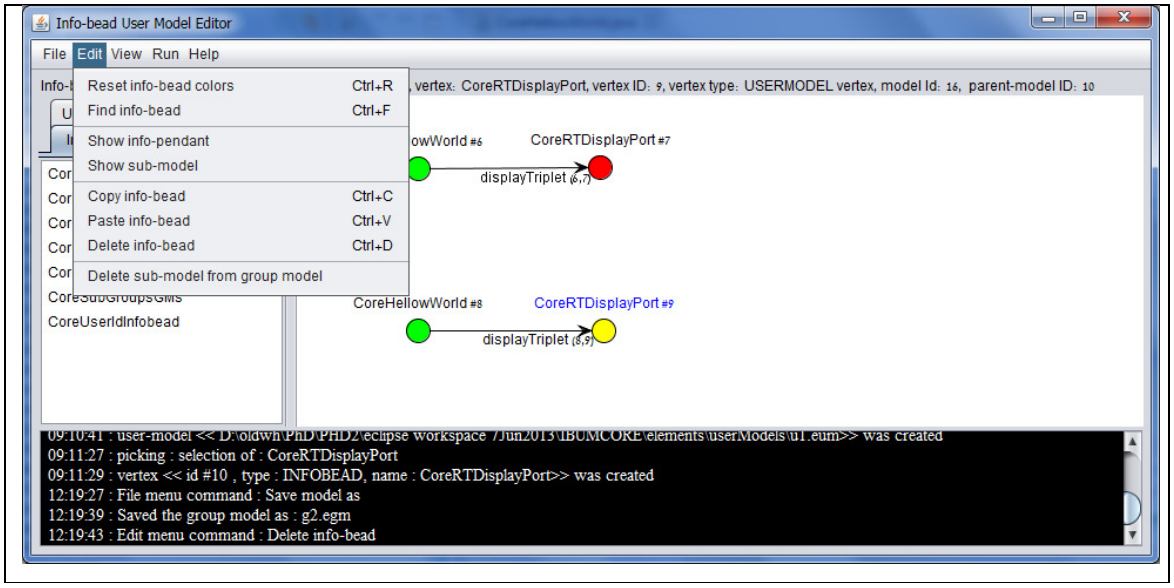

Fig. 12. The Edit menu

5.7.5 The View menu

The View menu is presented in Figure 13. It has the following menu items:

- Zoom in – Explains how to zoom into the graph in the working area using the mouse scroll down function
- Zoom out - Explains how to zoom out the graph in the working area using the mouse scroll up function
- Pan – Explains how to move the current view of the graph within the working area by holding the left mouse button pressed and dragging the graph.
- Rotate - Explains how to rotate the graph by holding the left mouse button and the Shift key pressed and dragging the graph.
- Move info-bead - Explains how to move the currently selected info-nead by holding the left mouse button and the Alt key pressed and dragging the graph.

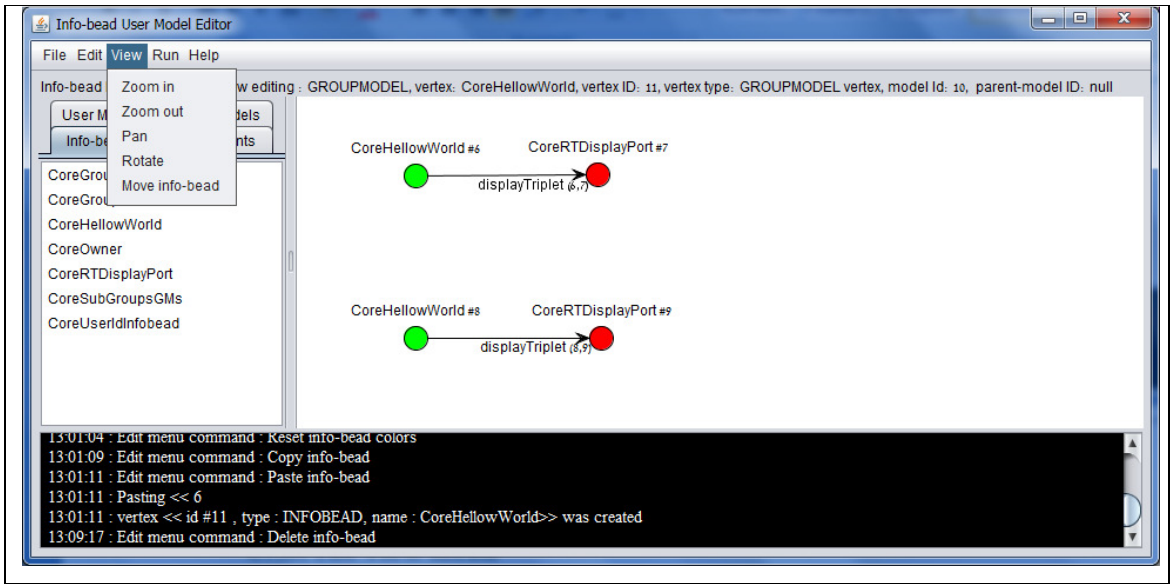

Fig. 13. The View menu

5.7.6 The Run menu

The Run menu is presented in Figure 14. It has a single menu item:

- Activate model – Automatically generates a model, that may run (i.e., be instantiated) by running the AcivatedModel1Main application (see section C4 above).

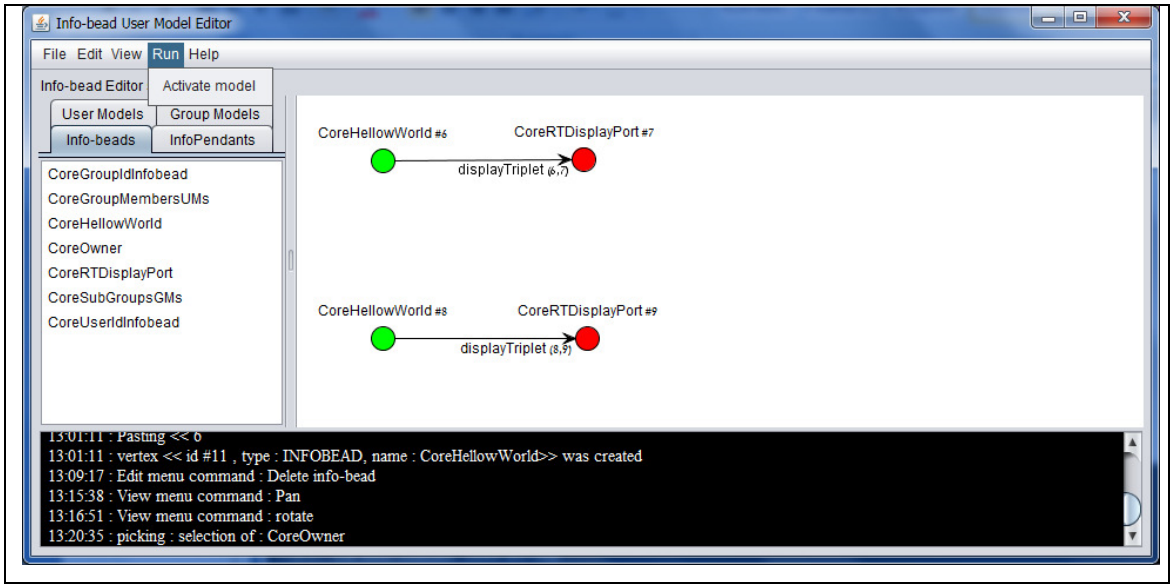

Fig. 14. The Run menu

5.7.7 The Help menu

The Help menu is presented in Figure 15. It has the following menu items:

- Introduction to info-bead user modeling – presents a link to an introduction document.
- Info-bead help – presents the content of the help file of the currently selected info-bead.
- About the Editor – Presents the editor configuration control and copyright information.

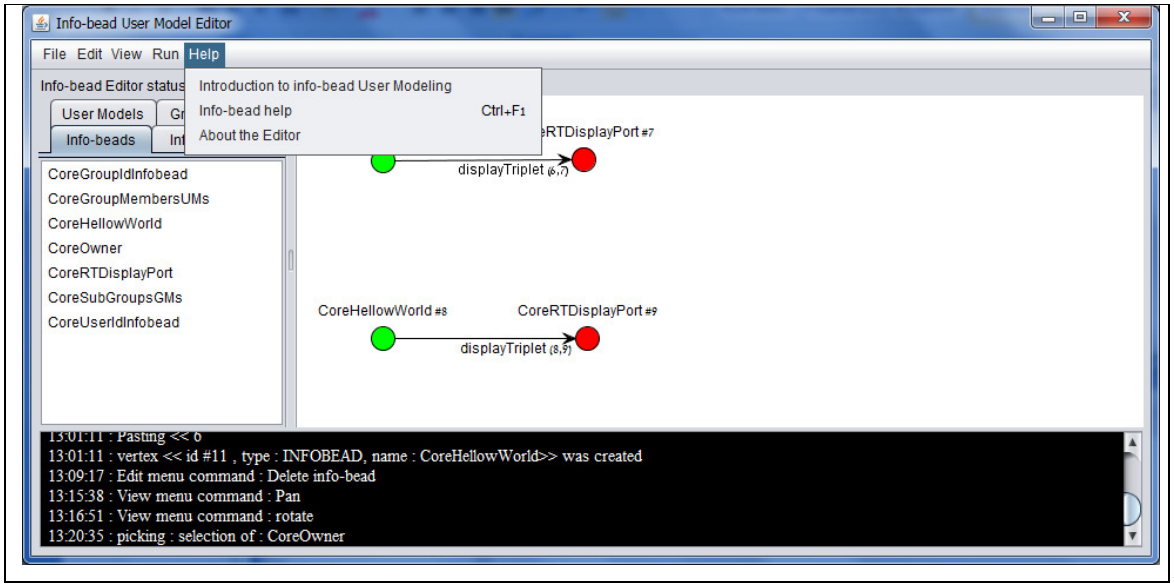

Fig. 15. The Help menu