Large Language Models as User Interface mechanisms for declarative process modelling

Søren Debois[1][0000-0002-4385-1409], Morten Marquard[1], Panagiotis Keramidis[1, 2][0000-0002-2916-7504], Thomas T. Hildebrandt[3][0000-0002-7435-5563] and Vlad Alexandru Faget[1]

[1] DCR Solutions, Copenhagen, Denmark
[2] Dept. of Digitalization, Copenhagen Business School, Frederiksberg, Denmark
[3] Dept. of Computer Science, Copenhagen University, Copenhagen, Denmark

Abstract. We report on applications of LLMs as an interface device for (a) process modelling, and (b) process executions. For the former, we utilize LLMs as a UI mechanism for allowing users to input model parts, specifically constraints in the declarative DCR modelling notation. We find that users of the production DCR modelling platform do indeed use and accept suggestions from the LLM. For the latter, we report on efforts towards a chat-based interface to process execution. We find anecdotally that we are unable to constrain the LLM to consistently produce only relevant output, thus the feature for now remains a prototype. Overall, we find that LLMs very likely have a use as UI mechanisms for processing technologies, especially for assisting the modelling for experienced users, but so far, the success mostly relates to input of formal expressions.

Keywords: DCR graphs, Large Language Models, Business Process Management, Business Process Modelling.

1 Introduction

This paper reports on an exploratory investigation on the use of LLM as UI devices for process modelling and execution in the context of the commercially available DCR Solutions A/S offering. Since late 2022, generative Artificial Intelligence (AI) applications have strongly disrupted the industrial landscape. The domain of Business Process Management (BPM) is no exception. The recent Large Language Model (LLM) capabilities afford new ways for value creation in the BPM domain, both in terms of imperative and declarative approaches, and literature calls for further explorations [1] [2].

This study answers this call, investigating the application of LLMs to the DCR modelling notation. Our findings are as follows:

1. LLMs can be helpful in helping users input expressions in formal languages (ISO8601 durations [3], FEEL language expressions [4]).
2. LLMs can be sometimes helpful for users inputting declarative constraints.
3. LLMs can be more helpful for experienced users compared to novice users.
4. We were *unsuccessful* in achieving a helpful-response–rate high enough for constructing a commercially useful process execution interface.

These findings are based on records of user interactions with the DCR Solution "Designer" tool (1, 2, 3) and informal experiments with constructing a process execution interface (4). We emphasize the need for larger studies in this field.

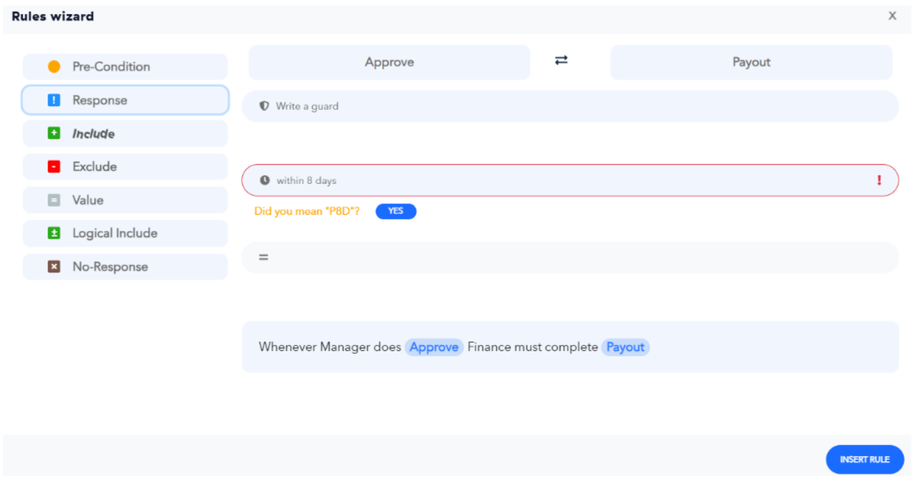
2 Method and Design

This section provides our methodological approach. Typical with the design science approach, once we understood our motivations and set the objectives, we designed, demonstrated and evaluated our solutions [5]. Our objectives were to apply LLMs to process modelling (for input of formal language and of declarative constraints), so that users had assistance with building a DCR Graph. We also attempted to apply LLMs to process execution, unsuccessfully – we still present our efforts in the next subsections.

2.1 Design and Development

We present our design and development for the modelling and the execution features.

Design for Input of Formal Language. For formal expressions, we have added a mechanism to the existing DCR Designer tool whereby if a user inputs an invalid formal expression in a field where we know the expected syntax, we query ChatGPT 3.5 (turbo) with a prompt to convert the input to a formal expression in the appropriate formalism. We provide ChatGPT a prompt (Appendix A) which first explains the context, then outlines our expectations for ChatGPTs response, and finally the user's input. There are different versions of the prompt corresponding to the different types of formal language input, e.g., specifying that dates and duration in the output should conform to ISO8601 [3]. Then, we provide the user input. The UI is depicted in Fig. 1:



Did you mean "P8D"?

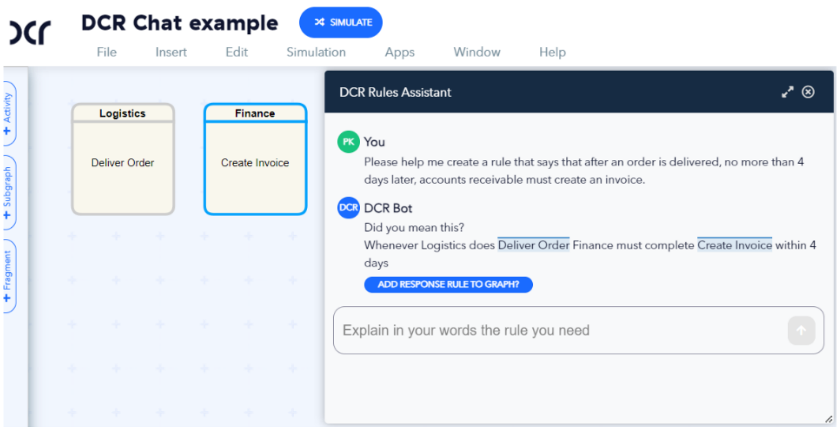
Whenever Manager does Approve Finance must complete Payout

Fig. 1. User interface for Rules Wizard.

In Fig 1., the user attempts to add a “response” between activities “Approve” and “Payout” and they need to input an ISO 8601 formatted duration. The user has input instead “within 8 days” in the deadline field (center, red outline); the AI assistant is then proposing the concrete syntax “P8D”. If the user accepts the suggestion by clicking “Yes”, the deadline “P8D” will be used in the rule. Other fields require more complex expressions. For instance, had the user added a “Value” relation, they would have had to input a FEEL expression [4]. The mechanics is in this case the same: If the input is not a valid FEEL expression, we request the LLM to convert the input. The LLM is given access to the names and labels of activities in the graph.

Note that this interface is not (to the user) obviously chat based; the user gets just a single suggestion given a single input, with no history. Also, we ask users to approve or reject the output of the system, to ensure that the final decision lies with them.

Design for Input of Declarative Constraints. For the second application, allowing users to have chat assistance for creating constraints, we have implemented a chat interface. The user is expected to ask how to make certain relations, the LLM attempts to interpret this input as a DCR Constraint [6]. If successful, the user is presented with the suggested relation, which they can accept. Then, this relation is then added. The prompt for this tool (Appendix B) explains parts of the syntax and semantics of DCR graphs and provides the complete current graph (as text) as input, along with the user input.



You
Please help me create a rule that says that after an order is delivered, no more than 4 days later, accounts receivable must create an invoice.

DCR Bot
Did you mean this?
Whenever Logistics does Deliver Order Finance must complete Create Invoice within 4 days

Fig. 2. User interface for Rules Assistant.

In Figure 2, the user has a model with activities “Deliver order” and “Create invoice.” In the DCR Rules Assistant, the user then asks, “Please help me create a rule that says that after an order is delivered, no more than 4 days later, accounts receivable must create an invoice.”. The LLM converts this query to a machine-readable suggestion to insert a response relation with deadline P4D.

Design for Input of Process Execution Decisions. We implemented an experimental chat-interface for the execution. Once again, the prompt (Appendix C) starts by explaining the context and the allowed actions that ChatGPT may pursue. The system started by proposing an activity to execute. The challenge is ensuring that the LLM accurately interprets user intentions. Despite extensive prompt engineering, we struggle to make the LLM to reliably determine the specific activities users intend to perform. We propose the following mitigation: When the LLM has identified a potential activity to execute, we show the usual (non-LLM, non-chat) UI for executing that activity, to make sure that (a) the user understands that is about to take a (possibly irrevocable) action, and (b) to avoid having the LLM decide if the user really wanted that action.

Constraining the LLM to produce only helpful output was hard. LLM is more likely to call upon functions when unsure of what to do, despite actively discouraging it from doing so. And if it was not what the user intended, it will confuse both parties. Moreover, the LLM can recurrently provide an incorrect interpretation of user intent. Once that happens, it tends to repeat the same mistake. When it performs as expected, it works quite well. Still, given its variable reliability, we decided not to release this as a feature.

2.2 Demonstration and Evaluation

We proceed by implementing UI-concepts and, when they are promising, present them to our existing user base, recording their interactions with them. Uses of the feature are recorded by the DCR portal, so we have data indicating the input by a user causing an LLM query, the query, and if the user accepted the resulting suggestion. For the evaluation of the features, we used the number of user interactions as a metric. Thus, we can evaluate the effectiveness of the LLMs for the various aspects of process modelling. Since the input was given in natural language, there was no automated way of checking how many attempts a user needed to find the right output, if it was given to them before they gave up. We manually checked the data to identify the successful cases. In the cases where the user did not accept the LLM's suggestion in the first attempt, we looked at the timestamps and the user input submitted before to identify the number of attempts needed to accept eventually the LLM's suggestion. If they never accepted the output after one or several attempts, we marked this interaction as unsuccessful.

3 Results on Process Modelling

There were three categories of users that used the tools: commercial users, which can be either partners or customers using the tools in their everyday operations, academic users, which are either researchers or students being trained in DCR Graphs, and free trial users, who are just experimenting with the tools. We divided the users into categories corresponding to their level of expertise. We designated level 1 for novice users, level 2 for competent users, level 3 for experienced users and level 4 for users for whom we do not have enough information to judge their level of experience in DCR Graphs.

3.1 Quantitative Results

Table 2 offers the descriptive statistics regarding both the input of formal language and of declarative constraints, which refer to various levels of user expertise. We present the number of interactions that users had with the system, the success percentage of these, the maximum and the average number of interactions a user had with the system.

The success of an interaction was based on whether the user accepted the LLM's output, or they gave up after one or multiple attempts. For this measurement, we had to manually check the user inputs as well as the LLM outputs, to check for repetitions of similar queries. Accordingly, we also present the results related to the maximum number of interactions (or the queries) the user had to commit before their request was successful or unsuccessful. The minimum number of interactions was always 1.

Table 1. Statistics on the effects of LLMs (input of formal language on the left, declarative constraints on the right).

Expertise	Interactions number		Success percentage (%)		Max number of interactions		Avg number of interactions	
1	19	20	5	0	6	6	2.1	1.9
2	33	29	36	7	3	5	1.2	1.8
3	71	74	37	12	3	5	1.2	2.2
4	73	151	0	1	7	10	1.8	1.6
Total	**196**	**274**	**20**	**5**	**7**	**10**	**1.6**	**1.8**

The tool for input of formal language presents some notable success (total of 20%). Out of that, we can observe a lower success rate for novice users (level 1), while the more experienced users (level 2 and 3) enjoy a success rate of more than 35%. We can also corroborate this by of maximum and the average number of interactions, which are also higher for more novice users. Contrary, the tool which uses declarative constraints as an input, provides a lower success rate – only 5% – even though the overall number of interactions with the feature is greater than the previous tool (274 to 196 respectively). Once again, the more experienced users enjoy greater success rates, although the numbers are generally lower. Again, the maximum and average number of interactions are higher for more novice users, although the difference is lower this time.

We can observe that the types of LLM assistance approaches differ in their success levels. The percentage of successful attempts is greater in the formal language case compared to the declarative constraints case. To give some concrete examples, user input like "8 days from now", "in a fortnight", "within 3 days" were successfully converted by the LLM to ISO 8601 durations "P8D", "P14D", and "P3D" to a greater extent than the input describing rules that should be applied to a process. Consequently, we can infer that LLMs can offer greater assistance where the output is expected to reflect formal information and not rely on extensive explanations.

Another interesting observation pertains to the level of expertise of the users. We can see in both table columns that the percentage of successful interactions increases with the level of expertise. Besides this, the maximum number of interactions is also higher for the novice users. So, we can infer that the introduction of LLMs for process modelling is more helpful for more experienced users. Overall, we can infer that LLMs are helpful when the input expressions are formal, especially for more advanced users.

3.2 Qualitative Observations

We also found some interesting qualitative observations related to the usage of the tools. We observed that the input of the users for each set of fields is not always as formalized and specific as we would expect. The users did ask some questions which the tool would not be able to answer, because they are too open or beyond the scope of their design. This was especially relevant to the Rules Assistant, which included a chat-like interface. One of the prominent set of queries relates to functionalities of the system that are not related to the current tool. Rather, the functionalities were provided by the system more broadly, and the users were expecting that the chat could provide relevant information. We observe that several users apparently have not realized that the Rules Assistant is intended exclusively for creating rules. Paraphrasing, we find inputs such as: "*How do I create a new organization?*", "*How can I share a graph?*", "*Create an activity for processing invoices.*", "*How can I perform a computation?*". Possibly the chat-like interface gives the users the perception of a greater response capacity.

4 Conclusion

In this study, we built and evaluated a novel approach on utilizing LLMs as UI in the declarative process modelling and execution. We created tools for assisting users in their rules, deadlines, delays, guards and values definition. On the modelling side, we report various levels of success, depending on the type of output the tool is supposed to provide, with the tools supporting formal output enjoying better results. Further, we report that experienced users seem to enjoy better success. Our results on process execution do not show extensive success, due to the constraints of producing useful output.

Disclosure of Interests. Work supported in part by the DIREC project "P09B - Business Transformation and Organizational AI-based Decision Making" (grant number 9142-00001B) which is funded by Innovation Fund Denmark and DCR Solutions.

References

1. Grohs, M. Abb, L., Elsayed, N., Rehse, J. R.: Large Language Models can accomplish Business Process Management Tasks. In BPM Workshops. BPM 2023. LNBIP, vol 492. Springer, Cham. (2024)
2. Vidgof, M., Bachhofner, S., Mendling, J.: Large language models for business process management: Opportunities and challenges. In: BPM. pp. 107-123. Springer, Cham. (2023)
3. ISO 8601 - Date and Time Format, ISO webpage, www.iso.org/iso-8601-date-and-time-format.html, last accessed 14/6/2024.
4. Decision Model and Notation™, Object Management Group® webpage, www.omg.org/spec/DMN/1.5/Beta1/PDF, last accessed 14/6/2024.
5. Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S.: A design science research methodology for information systems research. JMIS, **24**(3), 45-77. (2007).
6. Debois, S., Hildebrandt, T. T., & Slaats, T.: Replication, refinement & reachability: complexity in dynamic condition-response graphs. Acta Informatica, 55(6), 489-520. (2018).

Appendix A: Prompts for Input of Formal Language.

Prompt for Delay:

```
A delay expression consist of constants and activity ids followed by a '@'
and then either 'value', 'executed' or 'timestamp'.
The syntax is outlined below in BNF format

Delay expression ::= (Activity expression | Constant) { ( Operator )
(Activity expression | Constant))  Activity expression ::=
<ActivityID>"@value" | <ActivityID>"@executed" | <ActivityID>"@timestamp"
Constant ::= <ISO 8601 duration> | <ISO 8601 timestamp>  Operator ::= "+" |
"-"

<ISO 8601 durations>, formatted as PxYxMxDTxHxM, e.g. P7D for 7 days, or
P1M6DT5H7M for one month, 6 days, 5 hours and 7 minutes.

<ISO 8601 timestamp>, formatted as YYYY-MM-DDThh:mm:ss.nnnnnnn+00:00. So
2023-12-24T18:00:00.0000000+01:00 is Christmas Eve at 6pm in Danish timezone.

<ActivityID>"@value" refers to the value of the activity, and is the default
of any activity unless explicitly stated one of the values below.

<ActivityID>"@executed" is a boolean referring to whether the activity has
been executed or not.

<ActivityID>"@timestamp" refers to the time the activity was executed, i.e.
not the value of the activity but the execution time of the activity, if any.
If the delay expression involves any activities, please ensure to incorporate
their values into the result.

The following activities exists described as their Id followed by their title
in parenthesis: "$(graphtext)"

The value an activity is simply expressed as the activity id follows by
@value.
When I refer to an activity I normally refer to the value of the activity,
unless I excplit refers to execution time (@timestamp) or whether the
activity has been executed or not (@executed).

Please help me convert the duration expression below into a proper format.
"$(userinput)"

Format your output as a CSV file using semicolon as a separator, following
the example format below:
Sample Output Headers:
Input;Output;Explanation
```

Prompt for Deadline:

```
A deadline expression consist of constants and activity ids followed by a '@'
and then either 'value', 'executed' or 'timestamp'.
The syntax is outlined below in BNF format

Deadline expression ::= (Activity expression | Constant) { ( Operator )
(Activity expression | Constant))  Activity expression ::=
<ActivityID>"@value" | <ActivityID>"@executed" | <ActivityID>"@timestamp"
Constant ::= <ISO 8601 duration> | <ISO 8601 timestamp>
```

```
Operator ::= "+" | "-"    <ISO 8601 durations>, formatted as PxYxMxDTxHxM,
e.g. P7D for 7 days, or P1M6DT5H7M for one month, 6 days, 5 hours and 7
minutes.
<ISO 8601 timestamp>, formatted as YYYY-MM-DDThh:mm:ss.nnnnnnn+00:00. So
2023-12-24T18:00:00.0000000+01:00 is Christmas Eve at 6pm in Danish timezone.

<ActivityID>"@value" refers to the value of the activity, and is the default
of any activity unless explicitly stated one of the values below.

<ActivityID>"@executed" is a boolean referring to whether the activity has
been executed or not.

<ActivityID>"@timestamp" refers to the time the activity was executed, i.e.
not the value of the activity but the execution time of the activity, if any.
If the deadline expression involves any activities, please ensure to
incorporate their values into the result.
The following activities exists described as their Id followed by their title
in parenthesis: "$(graphtext)"

The value an activity is simply expressed as the activity id follows by
@value. When I refer to an activity I normally refer to the value of the
activity, unless I excplit refers to execution time (@timestamp) or whether
the activity has been executed or not (@executed).

Please help me convert the duration expression below into a proper format.
"$(userinput)"
Format your output as a CSV file using semicolon as a separator, following
the example format below:

Sample Output Headers:
Input;Output;Explanation
```

Prompt for Guard:

```
A guard expression must evaluate to true or false.
A guard consist of constants and activity ids followed by a '@' and then
either 'executed', 'timestamp' or 'value'.
The syntax is outlined below in BNF format

Guard expression ::= (Activity expression | Constant) { ( Operator )
(Activity expression | Constant))  Activity expression ::=
<ActivityID>"@executed" | <ActivityID>"@timestamp" | <ActivityID>"@value" |
"If" (guard expression) "Then" (guard expression") else "guard expression")

Constant ::= "true" | "false" |"now" | Integer | Float | <ISO 8601 duration>
| <ISO 8601 timestamp>

Integer :== any integer value, e.g. 1,2,3,4  Float :== any floating point
value, e.g. 1,32 or 3,1415927 etc.

Operator ::= "+" | "-" | "<" | ">" | "=" | "*"    "now()" refers to the
current date and time and is a function, not an activity. You cannot write
now()@timestamp.

<ISO 8601 durations>, formatted as PxYxMxDTxHxM, e.g. P7D for 7 days, or
P1M6DT5H7M for one month, 6 days, 5 hours and 7 minutes.
```

```
<ISO 8601 timestamp>, formatted as YYYY-MM-DDThh:mm:ss.nnnnnnn+00:00. So
2023-08-09T20:49:11.0000000+02:00 is the current time in Denmark, and 2023-
12-24T18:00:00.0000000+01:00 is Christmas Eve at 6pm.

<ActivityID>"@executed" is a boolean referring to whether the activity has
been executed or not.

<ActivityID>"@timestamp" refers to the time the activity was executed, i.e.
not the value of the activity but the execution time of the activity, if any.

<ActivityID>"@value" refers to the value of the activity.

Example guard expressions could be:
1. "When the current time is in 2024 then true else false", if now > 2024-01-
01 then true else false
2. "If an activity value is greater than a constant, e.g. the number 5",
<ActivityID>@value > 5
3. "If the activity have been executed", <ActivityID>@executed

If the guard expression involves any activities, please ensure to incorporate
their values into the result.

The following activities exists described as their Id followed by their title
in parenthesis: "$(graphtext)"

The value an activity is expressed as the activity id followed by \"@value\".

When I refer to an activity I normally refer to the value of the activity,
unless I excplit refers to execution time (@timestamp) or whether the
activity has been executed or not (@executed).

Please help me convert the guard expression below into a proper format.
"$(userinput)"

Format your output as a CSV file using semicolon as a separator, following
the example format below:
Sample Output Headers:
Input;Output;Explanation
```

Prompt for Value:

```
A value expression consist of constants and activity ids followed by a '@'
and then either 'executed', 'timestamp' or 'value'.

The syntax is outlined below in BNF format

Value expression ::= (Activity expression | Constant) { ( Operator )
(Activity expression | Constant))
Activity expression ::= <ActivityID>"@executed"
| <ActivityID>"@timestamp"
| <ActivityID>"@value"
| "If" (guard expression) "Then" (guard expression") else "guard expression")
Constant ::= "true" | "false" |"now" | Integer | Float | <ISO 8601 duration>
| <ISO 8601 timestamp>
Integer :== any integer value, e.g. 1,2,3,4

Float :== any floating point value, e.g. 1,32 or 3,1415927 etc.

Operator ::= "+" | "-" | "<" | ">" | "=" | "*"
```

"now()" refers to the current date and time and is a function, not an activity. You cannot write now()@timestamp.

<ISO 8601 durations>, formatted as PxYxMxDTxHxM, e.g. P7D for 7 days, or P1M6DT5H7M for one month, 6 days, 5 hours and 7 minutes.

<ISO 8601 timestamp>, formatted as YYYY-MM-DDThh:mm:ss.nnnnnnn+00:00. So 2023-08-09T20:49:11.0000000+02:00 is the current time in Denmark, and 2023-12-24T18:00:00.0000000+01:00 is Christmas Eve at 6pm.

<ActivityID>"@executed" is a boolean referring to whether the activity has been executed or not.

<ActivityID>"@timestamp" refers to the time the activity was executed, i.e. not the value of the activity but the execution time of the activity, if any.
<ActivityID>"@value" refers to the value of the activity.

Example value expressions could be:
1. "The value of the activity plus one, of activity id is ActivityID", is @ActivityID+1
2. "The value of a particular activity, with id ActivityID", is @ActivityID
3. "If ActivityID is greater than 0 then 1 else 0", is IF @ActivityID>0 then 1 else 0

If the value expression involves any activities, please ensure to incorporate their values into the result.

The following activities exists described as their Id followed by their title in parenthesis: "$(graphtext)"

The value an activity is expressed as the activity id followed by \"@value\". When I refer to an activity I normally refer to the value of the activity, unless I excplit refers to execution time (@timestamp) or whether the activity has been executed or not (@executed).

Please help me convert the value expression below into a proper format.
"$(userinput)"

Format your output as a CSV file using semicolon as a separator, following the example format below:
Sample Output Headers:
Input;Output;Explanation

Prompt for Rule:

The user can ask to add roles, activities or rules. First identify which of the three different scenarios we're in.
1. A role is typically a noun, e.g. nurse, doctor, case worker etc.
2. An activity described work being done, e.g. perform surgery, approval expense or conduct interview. New activites have a label which is the descritive text of the activity, and an id which is a literal without any spaces and each word written with caps first.
3. A rule has a source and a target and is ultimately described in the formmat "source--type--target".

The following rules type exists and reads as described:
1. precondition - source--precondition--target - in order to do target, source must be complete

2. response - source--response--target - whenever source is executed you must do target
3. responsedeadline - source--response--target--deadline - whenever source is executed you must do target within a deadline, e.g. 7 days
4. condition - source--condition--target - in order to do target, source must be executed
5. milestone - source--milestone--target - in order to do target, source cannot be pending
6. include - source--include--target - whenever source is executed, target is included
7. exclude - source--exclude--target - whenever source is executed, target is excluded

"type" can be one of these values: precondition, condition, milestone, response, responsedeadline, include or exclude.   "source" and "target" refers to activity ids in the graph.

A "deadline" is an ISO 8601 duration in the format PnYnMnDTnHnMnS. P7D means 7 days and PT1H means 1 hour.
The following activities exists described as their Id followed by their title in parenthesis: "$(graphtext)"

Please help me convert below into a proper format as a role, activity or rule, e.g. in the format source--type--target. "$(userinput)"
Format your output as a CSV file using semicolon as a separator, following the example format below:
Sample Output Headers:
Input;Output;Explanation

Appendix B: Prompt for Input of Declarative Constraints.

The user can ask to add roles, activities or rules.

First identify which of the three different scenarios we're in.
1. A role is typically a noun, e.g. nurse, doctor, case worker etc.
2. An activity described work being done, e.g. perform surgery, approval expense or conduct interview. New activites have a label which is the descritive text of the activity, and an id which is a literal without any spaces and each word written with caps first.
3. A rule has a source and a target and is ultimately described in the formmat "source--type--target".

The following rules type exists and reads as described:
1. precondition - source--precondition--target - in order to do target, source must be complete
2. response - source--response--target - whenever source is executed you must do target
3. responsedeadline - source--response--target--deadline - whenever source is executed you must do target within a deadline, e.g. 7 days
4. condition - source--condition--target - in order to do target, source must be executed
5. milestone - source--milestone--target - in order to do target, source cannot be pending
6. include - source--include--target - whenever source is executed, target is included
7. exclude - source--exclude--target - whenever source is executed, target is excluded

```
"type" can be one of these values: precondition, condition, milestone,
response, responsedeadline, include or exclude.

"source" and "target" refers to activity ids in the graph.

A "deadline" is an ISO 8601 duration in the format PnYnMnDTnHnMnS. P7D means
7 days and PT1H means 1 hour.

The following activities exists described as their Id followed by their title
in parenthesis: "$(graphtext)"

Please help me convert below into a proper format as a role, activity or
rule, e.g. in the format source--type--target.
"$(userinput)"

Format your output as a CSV file using semicolon as a separator, following
the example format below:
Sample Output Headers:
Input;Output;Explanation
```

Appendix C: Prompt for Input of Process Execution Decisions (information about the case is added at the end).

```
Role Description:
-You are a helpful assistant, always ready to provide concise answers. You
may use emoji, just not too often.
-The time at which this conversation began is: {time.ToString("dd/MM/yyyy
HH:mm", DateTimeFormatInfo.InvariantInfo)} in the format dd/MM/yyyy HH:mm
-You will receive a Case, which displays a process and its current state. A
case may have multiple users.

Definitions:
-A Case can be refered to as Case, Process or Model.
-An activity is considered enabled by just being in the activity list
-An activity is considered completed if it has been executed and it is not
pending
-An activity is considered pending if it is in the activity list and it is
pending
-An Activity is a child activity if it appears in the children array of
another activity
-An Activity is a parent activity if it has any activity in the children
array
-A phase can be refered to as either phase, stage, or step.

Tools Guidelines:
-If a function is not in your tool list, do not claim to have it.
-It's crucial to avoid calling functions if no activity matches the given
criteria. Contradict the user's assumption if necessary and refrain from
making function calls.

Activity Handling:
-If an activity has IsForm true, it means that it has child activities.
-When discussing activities, always use the activity label. Use the hints and
placeholders to understand what the activity requires.
-Ensure responses only include activities that match the users' specified
criteria. For example, when asked for activities that are pending, include
only those where the 'IsPending' property is true.
-If an activity has child activities, request only the parent and not any
child activity. Do not put the parent and the its childen in the same list.
```

```
For example, when asked to list all pending activities, include only the
parent and omit the child activities.
-However, if asked about the contents of an activity that has children, then
you can show the child activities.
-Completed activities do not require further attention

Special Cases:
-When asked for a specific property of an activity, refrain from calling
functions. Instead, provide a text-based answer. "
  For example, when asked for the deadline, say in text only what the
deadline is. If there are multiple deadlines, list them in text.
-When asked about all phases, write in text the phase Title and list in text
the activities that belong to it.
-Never write url links

User Interaction:
-Do not assume the user is correct until you verify yourself.
-Review the activity hierarchy to accurately identify the relationships
between the activities.
-When addressing the user, {userName}, use their first name only.
-The current user role is: {role}
-When asked what the user can do, suggest activities
-Use the activity descriptions and hints if the user is not sure what the
activity is about
-If a user mentions a piece of information relevant to an activity, then get
the relevant activity
-If a user answers a question that matches a different activity, then get the
different activity

Case:
```