Chapter 6

Low-Level Description of Element Types and Interfaces

The low-level diagram addresses the lack of an abstract conceptual framework in more detail. The framework, designed for software systems that use previous docking results, is independent of the programming language, toolset, or paradigm used. Using the detailed diagram of the framework can prevent the development of a tool from scratch if an existing tool can be reused. A tool can be described abstractly (drawn as a component of the diagram) and compared to other abstract descriptions of existing tools. Furthermore, if a description of another existing tool does not exist and a software engineer needs to determine whether the tool will fit the framework, it can be compared to an abstract description of an element type. This chapter shows the low-level view of the framework in the form of the diagrammatic, textual and formal descriptions of element types and interfaces.

6.1 Diagrammatic description of the framework

One should be able to represent any potential specific scenario that would use the framework, with this type of low-level detailed diagram. This diagram is a generic model of a system that uses a docking result repository, showing all element types and all possible interfaces between them. It is based on the Unified Modelling Language (UML [230]) Component diagram. The element types are drawn as components and the interfaces between them are the typical "provided" and "required" interface connections. It also features arrows showing the direction of the flow of data in the particular interface. The element types MDE, MDRR, AT, ADS, and DM were described in detail in Section 4.3.3.

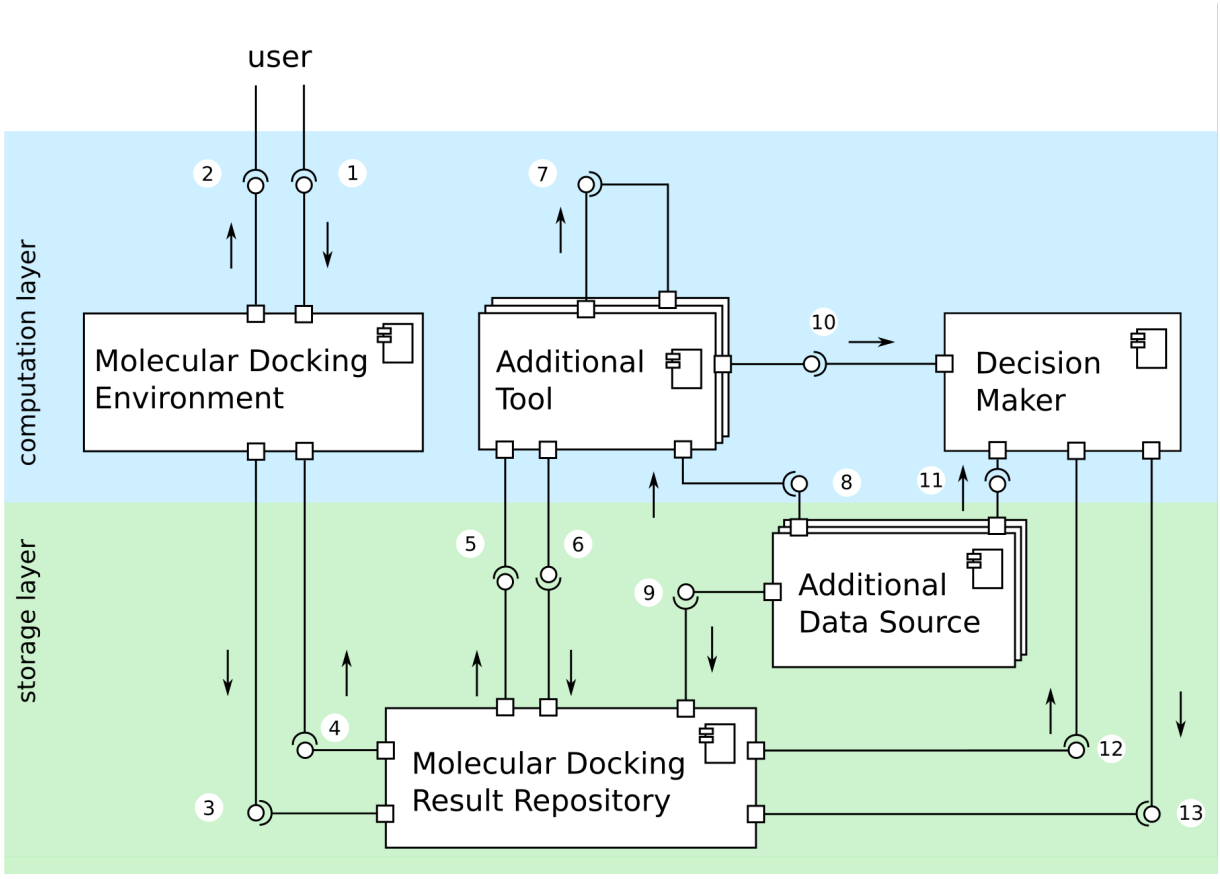

Figure 6.1: The diagram of the framework.

The framework has 13 interfaces between these element types (Figure 6.1):

1. user → MDE, provided by the MDE (since the user is not a true component): allows the user to upload the docking input or additional user input values needed by another element.
2. MDE → user, provided by the MDE: displays the result of the docking and other results from the MDRR to the user.
3. MDE → MDRR, provided by the MDE: allows the MDE to send docking results and other additional data that may be required.
4. MDRR → MDE, provided by the MDRR: allows the MDRR to send results of the analysis to the MDE.
5. MDRR → AT, provided by the MDRR: enables sending the appropriate input data to the AT.
6. AT → MDRR, provided by the AT: allows the AT to send results of the execution to the MDRR, in order to store them and keep track of the progress.
7. AT → AT, provided by the AT: allows one AT to send its results, or other required data, to another AT.

8. ADS → AT, provided by the ADS: enables querying the ADS for data, by the AT.
9. ADS → MDRR, provided by the ADS: querying the ADS for data, by the MDRR.
10. AT → DM, provided by the AT: allows an AT to send the results to the DM.
11. ADS → DM, provided by the ADS: enables querying the ADS for data, by the DM.
12. MDRR → DM, provided by the MDRR: allows the MDRR to send data to the DM.
13. DM → MDRR, provided by the DM: enables the DM to send results to the MDRR.

6.2 Textual description of element types and interfaces

MDE A tool that takes descriptions of molecules as input, runs molecular docking, and outputs the results. It can be a bundle of tools, which pre-process the description files, run the calculation, and process the results. It could contain a set of shell scripts, a workflow, or a set of workflows. The computing infrastructure it uses is completely independent of the MDE (Figure 6.2).

MDE interfaces

1. The MDE requires users to send files describing the ligands, receptors and the configuration file. The user sends the input files to the MDE. For example, Raccoon2 is an MDE for VS simulations, and it has a GUI that lets users upload a receptor, ligands and a config file. The user may send other data that is needed by another element (the user is not a software component, so this interface cannot be provided by the user).
2. The MDE provides an interface which displays the results back to the user.
3. The MDE provides an interface in order to deposit the results into an MDRR. All needed information should be sent including input and output (result) files.
4. The MDE needs to receive any analysis data or decision made directly from the MDRR. This requires an interface at the MDRR to send this data over.

MDRR A storage system where the molecular docking results are stored. It should also store meta-data so that the simulation from the MDE is reproducible. This can be a database with certain textual data as well as pointers to the path of files stored in a file system, or it could be a document-based NoSQL database where all the files are stored inside the database, or something similar (Figure 6.3).

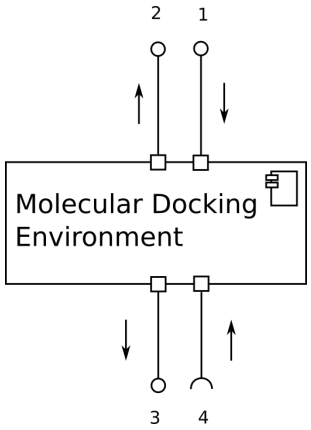

Figure 6.2: Diagram of the MDE.

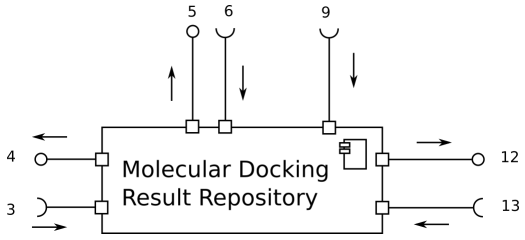

Figure 6.3: Diagram of the MDRR.

MDRR interfaces

3. The MDRR requires an interface to read the docking results and other data uploaded.
4. It provides an interface to send the result of the analysis to the MDE.
5. It provides an interface to send stored docking results data, in the needed format, to an AT.
6. It requires an interface to receive the AT results and keep a record of them.
9. The MDRR requires an interface in order to receive data from an ADS.
12. It provides an interface for sending stored docking results data in the needed format, to the DM.
13. Finally, it requires an interface to receive the AT results and keep a record of them.

AT The element type AT can be a tool that takes previous docking results from the MDRR as input, and runs another computation relevant to the particular scenario. It could take input data from a user sent through the MDE – MDRR, or from an ADS. It does not conduct docking simulations, but uses previous docking results or the input files for previous docking simulations. For instance, two receptors used in two docking experiments can be compared with a structural alignment tool. The AT can pass the results onto another AT, a DM, and it may send them to the MDRR (Figure 6.4).

AT interfaces

5. An AT (you can have multiple ATs) requires an interface that enables the MDRR to send stored docking results data so it can use them as input.

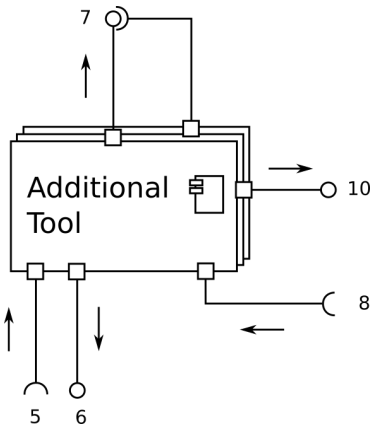

Figure 6.4: Diagram of the AT.

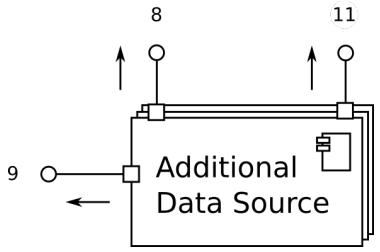

Figure 6.5: Diagram of the ADS.

6. An AT provides an interface that enables it to send its results to the MDRR to keep track of them.

7. An AT may provide an interface to send its results to another AT.

8. An AT may require an interface in order to obtain data from an ADS.

10. An AT provides an interface in order to send its results to the DM.

ADS The element type ADS represents a tool, such as a database, that stores relevant data. An ADS does not store the docking results, but other type of data that is additionally required in order to analyse previous docking results. It stores data that needs to be accessed by an AT, DM, or the MDRR. For instance, molecular properties about ligands can be read from an existing database. A scenario could read data from an external database, or include a copy of the database as part of the system (Figure 6.5).

ADS interfaces

8. An AT may need to use data stored in an ADS. The ADS should provide access to the data it stores, so that the AT can access it from its code.

9. It needs to provide access to its data for the MDRR as well.

11. Similarly, it needs to provide access to its data for the DM.

DM A tool (or bundle of tools) which make a decision based on results from an MDE and/or an AT. It gets the results directly from an AT, and can get more information from the MDRR. It may need to obtain additional information from the user (Figure 6.6).

DM interfaces

10. The DM requires an interface to receive results from one or more ATs.
11. The DM may require an interface defined at the ADS in order to obtain data from an ADS.
12. The DM requires an interface to receive previous docking results and other data from the MDRR.
13. The DM provides an interface to send the decision made to the MDRR.

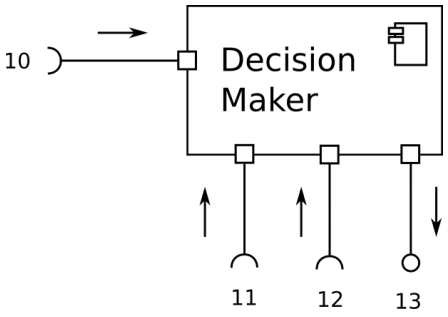

Figure 6.6: Diagram of the DM.

This concludes the low-level description of the framework with a detailed diagram and textual description of the element types and interfaces.

6.3 Formal description of element types and interfaces

To provide more objective means to compare an asbtract view of an existing tool with an element type, this section provides a formal description using Z notation (the rationale for using Z was outlined in Chapter 2). Using formal methods in the field of molecular docking is limited to research efforts such as [231], which does not utilise the popular *Z notation*. The formal description of element types and their interfaces has been written in CZT Eclipse [232], which automatically checks that the code conforms to the Z syntax. The full formal description of the framework is provided in Appendix B. It begins with a freetype and set definitions, assuming that there is a set *CHAR* which represents allowed characters. Regardless of the format that the input and output files use, they can be viewed as containing strings of characters. Multiple files are modelled using the set operator ($\mathbb{P}$). A mapping of the tuple ligand-receptor-config-date to a docking result is modelled as a previous docking result, *PREVIOUS_RESULT* (page 162).

6.3.1 Element types

MDE An MDE enables the execution of a docking simulation. A docking process requires a ligand and a receptor as input, and produces a docking result as output. Depending on the docking tool or use case, it may or may not require a configuration file (or configuration parameters). This is modelled by *dockingWithoutConfig*, *dockingWithConfig*, *Docking*, and *MolecularDockingEnvironment* in Appendix B, pages 162 - 163. An excerpt of the formal description showing the segment related to the MDE is shown in Figure 6.7.

$$dockingWithoutConfig : (LIGAND \times RECEPTOR) \rightarrowtail RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists res : RESULT \bullet dockingWithoutConfig(l, r) = res$$

$$dockingWithConfig : (LIGAND \times RECEPTOR \times CONFIG) \rightarrowtail RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists c : CONFIG;\ res : RESULT \mid c \neq \varnothing \bullet dockingWithConfig(l, r, c) = res$$

Docking

$ligand? : LIGAND$
$receptor? : RECEPTOR$
$config? : CONFIG$
$result! : RESULT$

$$config? = \varnothing \land result! = dockingWithoutConfig(ligand?, receptor?) \lor$$
$$config? \neq \varnothing \land result! = dockingWithConfig(ligand?, receptor?, config?)$$

MolecularDockingEnvironment

$ligands? : LIGANDS$
$receptors? : RECEPTORS$
$config? : CONFIG$
$results! : RESULTS$
$date! : DATE$

$$\exists ligand? : ligands?;\ receptor? : receptors?;\ result! : results! \bullet Docking$$

ViewMolecularDockingResults

$\Xi MolecularDockingEnvironment$

$$results! \neq \varnothing$$

Figure 6.7: Excerpt of the Z notation describing the MDE element type.

MDRR The MDRR should store data about the relation between a ligand-receptor-config-date tuple, and a docking result. It may include the decision made by a DM. The model is a minimal MDRR, acknowledging that some scenarios may require storing additional data such as author or version of docking tool used. They could be defined in a similar way to ligand or receptor, and included in the definition of *repository*. This would be reflected in all the interfaces to and from the MDRR. However, to compare a formal description of an existing tool to this abstract description of an MDRR, only the minimal data stored is required, as modelled in *MolecularDockingResultRepository*, Appendix B, page 164.

AT The AT is perhaps the most generic element type. The only restriction on the calculation it provides is that it is not another docking simulation. Different sub-types of ATs can be defined based on the type of input. Thus, an AT can use previous docking results (from an MDRR), data source information (from an ADS), additional tool result (from another AT), or a combination of them (which may also include user input). This is modelled by the axiomatic definitions in Appendix B, pages 165 - 168. The fact that an AT is defined by one of these types is shown in the schema *AdditionalTool*.

ADS The ADS is also modelled in a very generic manner. It is a database that stores data which is relevant for the system. This is modelled as a relation between a generic data source input and a resulting data source information. The very simple *Additional-DataSource* in Appendix B, page 169 shows this.

DM The purpose of the DM is to summarise the results it has received from ATs, the MDRR, or the user. The way that sub-types of DMs have been identified is similar to the identification of sub-types of ATs. Based on the combinations of inputs, there are several sub-types of DMs as shown in Appendix B, pages 169 - 171. The fact that a DM is defined by one of these types is shown in the schema *DecisionMaker*.

6.3.2 Interfaces

Interface 1: User, MDE This interface is modelled by specifying the user-provided text files as input variables (using the suffix "*?*"), Appendix B, page 163.

Interface 2: MDE, User This interface is modelled by a schema showing that the docking results can be viewed by the user, as long as they exist and they are output variables (suffix "*!*"), Appendix B, page 163.

Interface 3: MDE, MDRR An MDE can insert one or more docking results, as modelled by two schemas in Appendix B, page 164. The Δ operator signifies that there will be a change, which is detailed by the override operator ($\oplus$). A new item will be inserted, unless the ligand-receptor-config-date tuple is the same as an existing one, in which case the MDRR will be updated.

Interface 4: MDRR, MDE Interfaces 4, 5, and 12 represent the selection of data from the MDRR into the MDE, AT, or DM respectively. The repository is modelled as a relation, so the appropriate domain ($\lhd$) and range restriction ($\rhd$) operators are used. The former is used to select a tuple based on the left-hand side of the relation (when the ligand, receptor, config, or date is known), and the latter based on the right-hand side (when the docking result is known). A total of 32 different combinations of selection types are outlined in the schema *SelectMolecularDockingResults*, Appendix B, page 165. The same schema models interfaces 4, 5, and 12.

Interface 5: MDRR, AT Please see description of Interface 4. Interface 5 is further described in *AdditionalTool* by the input variables *previousDockingResults* and *userInput* (which would be passed via the MDRR).

Interface 6: AT, MDRR This interface can be modelled similarly to the way that Interface 3 models the MDE inserting docking results into the MDRR. In order to do this, the formal description of the MDRR would need to include results from an AT.

Interface 7: AT, AT Additional tool results of another AT are defined as an input variable in the *AdditionalTool* schema showing that an AT can receive results of another AT. This action is described in more details in *ReadAnotherAdditionalToolResults* (Appendix B, pages 168 - 169).

Interface 8: ADS, AT An AT can use data from the ADS by selecting it accordingly. This action is modelled by the schema *SelectAdditionalDataInfo* which shows that data from the ADS can be selected (Appendix B, page 169). The right-hand side of the repository (data source info) is selected based on the value of the left-hand side (data source input). The domain restriction ($\lhd$) is used to select the items stored in the ADS. The range function, *ran()* is used to obtain the right-hand side values for the selected items.

Interface 9: ADS, MDRR This interface can be modelled similarly to the way that Interface 3 models the MDE inserting docking results into the MDRR. In order to do this, the formal description of the MDRR would need to explicitly include results from an ADS. The schema *SelectAdditionalDataInfo* shows that data from the ADS can be selected (Appendix B, page 169).

Interface 10: AT, DM The fact that the *additionalToolResult* is defined as an input variable in *DecisionMaker* (Appendix B, 171) is sufficient to model an interface between an AT and DM. If a sub-type of DM requires results from an AT as input, they can be received.

Interface 11: ADS, DM Please see Interface 8. The same schema is used to model the interface that the DM uses to select data from the ADS.

Interface 12: MDRR, DM Please see description of Interface 4. Interface 12 is further described by the fact that the *userInput* variable in *DecisionMaker* is an input variable (which would be passed through the MDRR).

Interface 13: DM, MDRR A new decision from the DM can be inserted into the MDRR, or an existing one can be updated. This is modelled with the help of the override operator ($\oplus$) in *InsertUpdateDecisionRepository*, Appendix B, page 164.

6.4 Conclusion

This chapter proposed an abstract conceptual framework which is independent of the programming language, toolset, or paradigm used by a software system. The framework can be used to describe systems that use previous molecular docking results. It provides three main functionalities. Firstly, a scenario can be described using the basic diagram of the framework in order to determine whether it could be implemented using the framework. Secondly, the use of the framework will create a library of abstract descriptions of existing tools. When seeking an existing tool to use in a system, the abstract description of the element type can be compared to the library to find a candidate existing tool. Thirdly, it includes abstract descriptions of generic element types. An existing tool can be described in the same format and compared to the appropriate element type, to find out if it can be used in an implementation of a system. The three uses of the framework are further described as the techniques of the methodology in Chapter 7.