Building Science Gateways for Analysing Molecular Docking Results Using a Generic Framework and Methodology

Damjan Temelkovski[1], Tamas Kiss, Gabor Terstyanszky, Pamela Greenwell

University of Westminster, 115 New Cavendish Street, London, W1W 6UW

Abstract

Molecular docking and virtual screening experiments require large computational and data resources and high-level user interfaces in the form of science gateways. While science gateways supporting such experiments are relatively common, there is a clearly identified need to design and implement more complex environments for further analysis of docking results. This paper describes a generic framework and a related methodology that supports the efficient development of such environments. The framework is modular enabling the reuse of already existing components. The methodology, which proposes three techniques that the development team can use, is agile and encourages active participation of end-users. Based on the framework and methodology, two prototype implementations of science-gateway-based docking environments are presented and evaluated. The first system recommends a receptor-ligand pair for the next docking experiment, and the second filters docking results based on ligand properties.

Keywords: bioinformatics; molecular docking; science gateway; virtual screening; Distributed Computing Infrastructure; cloud computing.

1. Introduction

Molecular docking (often simply "docking") is a computational simulation that models biochemical interactions to predict where and how two molecules would bind. A molecule is the smallest group of atoms that retain uniform biochemical properties. This paper focuses on two types of molecules: ligands - small molecules that can bind to other molecules, and receptors - large molecules that cause a biological change in a cell when a ligand is bound to them. Receptors are often proteins, polypeptides responsible for most functional and structural features in living organisms, made of a chain or a sequence of amino acids [1].

Large-scale docking simulations are used in areas such as drug discovery where they can decrease the amount of laboratory (wet-lab) experiments required to obtain a molecule that is a candidate drug. Large-scale docking of many ligands and one receptor is called virtual screening (VS). A typical VS experiment combines hundreds of thousands of docking simulations and is very computationally demanding, requiring the use of Distributed Computing Infrastructures (DCIs). Utilising and accessing such computational resources add an extra level of complexity to this task, making it increasingly difficult for biomedical scientists. Science gateways are widely used in this area to help bridge this gap. However, there is still a need for more complex environments that enable scientists to access a wide range of computing, data and network resources for the further analysis of docking results. These environments must enable scenarios where intelligent support can be provided for more efficient analysis of results of large-scale docking experiments.

This paper investigates such scenarios and proposes a generic conceptual framework and a related methodology that uses regular input from scientists to support the development of complex science-gateway-based environments for the storage, analysis and reuse of molecular docking results. Further introduction to this topic and a detailed problem statement can also be found in our previous work [2].

The proposed generic conceptual framework has been developed considering requirements collected from semi-structured interviews with biomedical scientists and a literature review of 14 related projects, listed in Section 2. From this generic framework, specific architectures can be derived supporting various molecular-docking-related analytical

[1] *Corresponding author*
Email addresses: damjan.temelkovski@my.westminster.ac.uk (Damjan Temelkovski), t.kiss@westminster.ac.uk (Tamas Kiss), g.terstyanszky@westminster.ac.uk (Gabor Terstyanszky), p.greenwell@westminster.ac.uk (Pamela Greenwell)

23 Feb, 2020

scenarios, as shown in Section 3. A software development methodology that supports creating software systems that use this framework is explained in Section 4. The methodology provides three techniques that can help the software development team determine which specific elements can be used in a scenario. In Section 5, the detailed design of two scenarios, based on the framework and methodology are presented followed by their implementation. This illustrates the utilisation of the three suggested techniques, and the benefits provided for the software developers, the primary beneficiaries. Usability tests in Section 6 broadly examine benefits of these implementations for the end-user, the biomedical scientist. Finally, Section 7 concludes the paper and outlines future work.

2. Related Work

The pioneering docking tool was first published in the 1980s. It was called DOCK [3] and considered both ligand and protein as rigid. One of its latest versions, DOCK6 [4], considers the flexibility of molecules too. Today there are over 50 available docking tools [5]. Proprietary docking tools such as GOLD [6] or FlexX [7] are an option, but AutoDock **Error! Reference source not found.** is reportedly the most popular docking tool based on number of citations [9]. An updated algorithm called AutoDock Vina **Error! Reference source not found.** is known to be faster and more accurate in certain cases [10]. In order to conduct a VS simulation, the user can write a custom script to run a docking tool multiple times. Alternatively, a VS tool with a graphical user interface (GUI) can also be used. Raccoon2 [12] is a desktop application for running VS simulations using AutoDock Vina. It provides a server connection manager for submitting jobs to a High-Performance Computing (HPC) cluster directly from the Raccoon2 GUI. Raccoon2 supports only clusters with the Portable Batch System (PBS) or Sun Grid Engine (SGE) schedulers.

While docking is used to estimate the binding affinity between a ligand and receptor, structural alignment is used to find similar receptors based on their structure (the three-dimensional position of atoms). Receptors can be compared by using their amino acid sequences in what is known as sequence alignment, but to understand its function or mechanisms of action, one needs to compare their three-dimensional structure. Based on available publications, it has been estimated that the number of new structural alignment tools doubles every 5 years [13]. The number of available structural alignment tools is over 100 [13-15]. The first major structural alignment tools started appearing in the 1990s with examples such as DALI [16] and CE [17]. A more recent tool that was top-ranked tool at the Critical Assessment of protein Structure Prediction (CASP) competitions CASP12 and CASP13, is RaptorX Structure Alignment server which uses the standalone tool DeepAlign [18].

Given the vast number of docking or structural alignment tools to choose from, a modular component-based approach to developing software environments can be beneficial. In such an approach, a component is implemented using interfaces and a core unit. If the interfaces are compatible, the core unit may be seamlessly replaced. Furthermore, instead of designing and coding the environment from scratch, a component-based approach would focus around assembling and reusing components. Component-based software environments can be modelled using the standardised Component Diagram in Unified Modeling Language (UML) 2 [19]. This enables a high-level graphical model that documents the details of the environment, such as the components, their interfaces, and their interactions. Areas such as systems engineering can benefit from domain-specific modelling languages including sysML [20] or EAST-ADL [21]. Other methods that can be used to model component-based software environments rely on textual representation of formulas known as formal languages. Examples include the Vienna Definition Language (VDL) [22], Alloy [23] or Z notation [24]. A state-based formal language inspired by mathematical set theory, Z notation allows grouping of formal rules in “schemas” that can contain state variables while logical and discrete mathematics expressions describe systems using mathematical conventions.

In conjunction with component-based approaches, environments that use molecular docking simulations are often developed as scientific workflows. Scientific workflow management systems enable users to run simulations on DCIs without any low-level programming, by merely specifying a workflow graph (where each node contains an executable, input, and output ports, and can be connected to other workflow nodes), and utilising built-in connectors to various DCIs. Scientific workflow engines that have been used for bioinformatics simulations such as VS include Kepler [25], Nextflow [26], Taverna [27], and WS-PGRADE/gUSE [28]. Workflows can be created graphically by drawing the nodes and edges, or textually by writing workflow description code. A workflow can be reused as a node in another workflow, i.e. it can be a sub-workflow.

Existing environments for VS can be divided into VS pipelines which contain a set of scripts or tools to be used in a particular order, and workflow-based docking systems which automate the preparation, execution, and analysis of docking experiments using scientific workflows.

In a specific example [29], a pipeline has been used to explore the off-target activity of the drug Nelfinavir using reverse docking (many receptors, one ligand) as well as rescoring (reanalysis of top scored ligands using molecular mechanics methods). Other drug discovery pipelines composed of docking and rescoring, such as [30], have been run on HPC clusters. Clusters have also been used to run VS pipelines composed of a set of Perl scripts in [31] and [32]. In the former, top-ranking results of the docking are stored in the file system, while the latter includes a MySQL database for storing input and output files. Clusters are also used as part of the web solution in [33]. Cloud computing has also been used for docking simulations in [34], where a small desktop application has been developed to submit jobs on the cloud. Glaab [35] provides a review of current open-source programs that can be used in a VS pipeline, split into different elements, along with a rudimentary Docker container where selected tools are deployed.

One example of a scientific workflow is WS-PGRADE and the gUSE services [36-38] which have been used to run docking experiments using various DCIs, such as clouds [39], grids [40] or HPC clusters [41]. In the latter example, interviews with one expert biochemist have been conducted to gather user requirements and implement a docking gateway. In all three examples, the results of the workflows are stored as part of the workflow management system, without additional metadata. On the other hand, in [42], a specific XML-based format has been created in order to store metadata about the execution, input, and output from three different molecular simulation domains, including docking. The workflows themselves can be stored in workflow repositories, such as myExperiment [43], or SHIWA **Error! Reference source not found.**.

Additionally, there are many similar systems that implement bioinformatic simulations that are different from docking. In one example [45], a pipeline featuring multiple bioinformatic steps including prediction of a template ligand that would bind to a target receptor and searching for similar ligands is presented. This pipeline includes additional steps to combine and rank the results of the ligand similarity. Several other examples create portal-based systems for related analysis. For instance, a large system with 19 domains, including receptor-receptor docking (which is different from receptor-ligand docking) [46]. Smaller scale portals focus on one domain, such as [47], which focuses on molecular dynamics. Finally, many portals for bioinformatics use workflow-based technologies, such as the proteomics data analysis portal shown in [48].

Related system	**Type**	**Stores simulation results**	**Additional analysis**
Xie et al. [29]	VS pipeline	Y	Y
Zhang et al. [30]	VS pipeline	Y	Y
DOVIS 2.0 [31]	VS pipeline	Y	Y
D'Ursi et al. [32]	VS pipeline	Y	Y
HADDOCK 2.2 [33]	VS pipeline	Y	
Kiss et al. [34]	VS pipeline	Y	
Glaab [35]	VS pipeline		
Farkas et al. [36]	Workflow-based	Y	
Kiss et al. [40]	Workflow-based	Y	Y
Jaghoori et al. [41]	Workflow-based	Y	
MoSGrid [42]	Workflow-based	Y	Y
Roy et al. [45]	Docking-equivalent		
WeNMR [46]	Docking-equivalent	Y	
GridMACS [47]	Docking-equivalent	Y	
Kunszt et al [48]	Docking-equivalent	Y	Y

Table 1 – Related existing environments for VS divided according to type, whether they provide storage for simulation results, and whether they enable additional analysis of the results

As shown in Table 1, some of the systems listed above do not store the simulation results and are mainly useful to provide the results directly to the user [35, 45]. Others store the results, but do not provide any additional analysis methods [33-34, 36, 46-47]. The remaining examples [29-32, 40, 42, 48] support additional analysis of the user's own simulation results only, and do not provide the functionality to analyse results produced by other users.

In this paper, we have gathered the specific types of components used in these 14 existing systems, as well as novel systems proposed through a series of interviews with five domain experts, in order to create a generic conceptual framework for systems that store docking results and provide additional analysis methods. The aim was to formalise

and speed up the development of such environments. To the best of our knowledge, when developing the systems shown in this section, software engineers used general software development methodologies which do not allow systematic analysis and reuse of existing building blocks. The methodology presented in this paper provides a formal manner of selecting components and determining whether they can be reused or whether a new component is suitable for the intended purpose.

3. Generic Framework for the Analysis of Molecular Docking Results

The aim of the presented research was to identify potential similarities in the work of biomedical scientists working with molecular docking experiments, and to investigate whether a generic framework for such application scenarios can be defined. The assumption was that based on this generic framework the development of specific science-gateway-based environments supporting various molecular docking scenarios and the analysis of results can be formalized and speeded up.

In order to identify typical user requirements, several interviews with five scientists from different backgrounds and with various degrees of experience with molecular docking simulations were conducted. This number of London-based interviewees was useful in producing several conclusions. The interviews aimed at identifying requirements of the scientists when performing molecular docking experiments and specifying scenarios that are not supported by currently available science gateways for molecular docking. These scenarios typically represent software systems that make a decision based on the docking results, mimicking the steps that a scientist needs to take after obtaining the results. Some representative and identified scenarios are listed below:

Scenario 1. Suggest a receptor-ligand pair that should be used in the next molecular docking, based on receptor similarity and previous results.

Scenario 2. Filter docking results suitable for wet-lab experiments, based on ligand properties.

Scenario 3. Find off-target drugs, based on deducing if the estimated binding is at an active site of the receptor.

Scenario 4. Enable verification of the docking methodology and learning from previous docking for novice users.

Scenario 5. Compare results from different molecular docking tools.

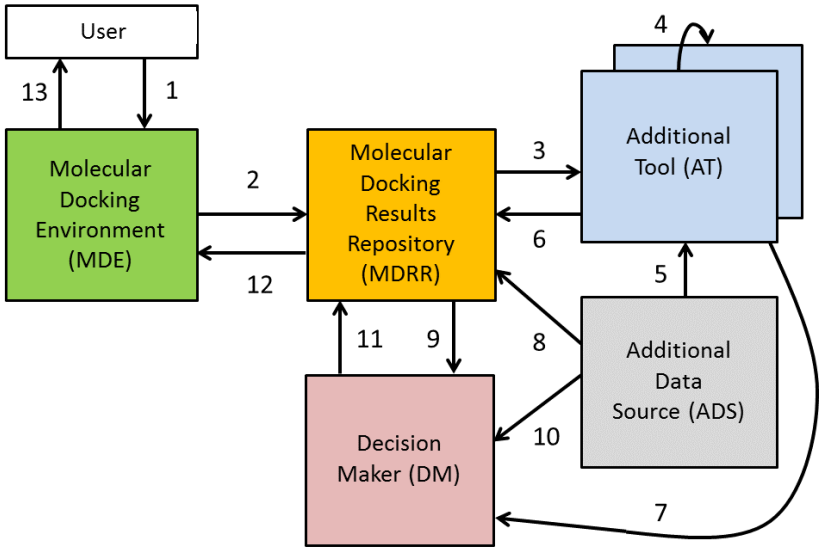

Figure 1 – Basic diagram of the framework - arrows show possible dataflow connections between element types. A user uploads input to the MDE (1) and docking data may be sent to the MDRR (2) which sends this or previous docking data to ATs (3). An AT may communicate with other ATs (4) or look up data in an ADS (5). An AT may supply its calculation results to the MDRR for storage (6) and to the DM (7). The MDRR may receive data from an ADS (8) and the DM may use data from the MDRR (9) or an ADS (10). The decision can be passed back to the MDRR for storage (11) and finally to the MDE (12) which provides visualisation (13).

Based on the conceptual similarities of these scenarios and an extended review of literature including the 14 papers referenced in Section 2, a generic framework has been designed. The design focuses on the similar elements in the

scenarios and includes the following element types (Figure 1):

Molecular Docking Environment (MDE): All scenarios include an environment where the molecular docking simulation is executed. It could be as simple as running a single simulation from the command line on a local computer, to more complex such as executing a virtual screening experiment on a DCI. The MDE includes the software tool used for the docking itself, it may be connected to a DCI, or include a high-level user interface.

Molecular Docking Results Repository (MDRR): After the execution of the molecular docking, the results and the input files need to be stored because previous docking results are useful for various scenarios. The repository shall also store information about the final decision made in the scenario.

Additional Tool (AT): The results which have been stored in the MDRR are then processed by an AT. This is a generic element that describes a tool which takes data from the MDRR as input and conducts a calculation. ATs can refer back to other data stored in the MDRR, communicate with other ATs, or use data stored in an Additional Data Source.

Additional Data Source (ADS): An ADS, such as an external database, shall contain data that is relevant for the final decision but is not docking result data or data referring to a docking simulation.

Decision Maker (DM): All the information processed from the different ATs and the ADSs would be passed to a DM. This element groups and analyses the performed calculations in order to make a decision.

The flow of data through the different element types is shown by the numbers in Figure 1.

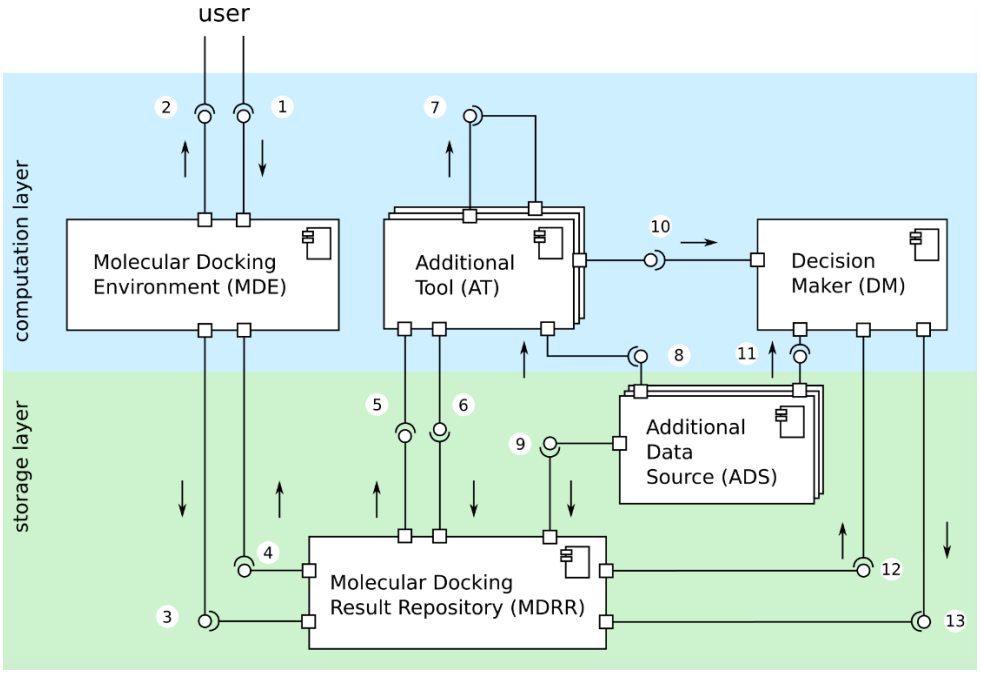

Figure 2 – Detailed diagram of the generic framework showing the location of needed interfaces. UML notation used as follows: rectangle with component symbol signifies "component", small square - interaction "port", stick with full white circle - "provided interface", stick with black crescent - "required interface", arrows show dataflow direction.

A more detailed view of the framework consisting of a detailed diagram, a textual description of elements and interfaces, and a formal description using Z notation, has been developed. The framework aims to provide a formalised way to derive specific scenarios, independent from the software engineering details of the implementation. The detailed diagram representing the framework in Figure 2 is a generic model, showing all generic element types and all possible types of interfaces between them. It is based on the UML Component Diagram with element types drawn as components and the types of interfaces as the typical *provided* and *required* interface connections. Additionally, it features arrows pointing towards the direction of the flow of data in a particular interface.

The framework features 13 types of interfaces. Each of these interfaces have been identified and described with a textual description. Representative examples are provided here. For example:

Interface 3: MDE → MDRR, provided by the MDE: allows the MDE to send docking results and additional data.

Interface 4: MDRR → MDE, provided by the MDRR: allows the MDRR to send analysis results to the MDE.

Following this, each element and each interface have been described formally using Z notation. A representative example of the formal description is presented here, describing the MDE and its interfaces (Figures 3 and 4). Please

note that the complete description of the generic framework, including both the textual description and the Z notation can be found at [49].

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

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

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

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

Docking

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

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

Figure 3 – Abstract formal description of MDE using Z notation. The core computation maps the input pair LIGAND-RECEPTOR or triple LIGAND-RECEPTOR-CONFIG into the docking RESULT in the functions dockingWithoutConfig() and dockingWithConfig() respectively. LIGAND, RECEPTOR, CONFIG, and RESULT have been defined as Z free types which describe the docking input/output files as sets of characters. Common mathematical symbols include: partial injective function ($\rightarrowtail$), input variable (suffix ?), and output variable (suffix !)

The docking process expressed by the MDE needs a ligand, receptor, and optionally configuration (config) files as input, and provides a docking result file as output. The Z notation (Figure 3) shows that, depending on the presence of a config file, the functions *dockingWithoutConfig()* or *dockingWithConfig()* will generate the docking result. They describe that for every non-empty ligand×receptor pair, there exists a docking result.

MolecularDockingEnvironment

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

ViewMolecularDockingResults

$$\Xi MolecularDockingEnvironment$$

$$results! \neq \varnothing$$

Figure 4 – Interface 1 described with Z notation using the input variables *ligands?*, *receptors?*, *config?*. Interfaces 2 and 3 described using the output variables *results!* and *date!*, with the condition that the results cannot be empty. LIGANDS defined as a set of LIGAND, RECEPTORS as a set of RECEPTOR, RESULTS as a set of RESULT. DATE defined as a set of characters describing a calendar date. Z symbols include Ξ to indicate no change of state.

Figure 4 models the MDE and interfaces 1, 2, and 3. This schema explains that the ligand, receptor, and config files are expected as input, while the docking results as well as information about the date are produced as output. The lower part of Figure 4 describes the interface that enables users to view results, as long as they exist.

The generic framework presented in this section was described using the basic diagram, the detailed diagram, the textual description and the Z notation. It will be shown in Section 5 that this generic framework can be used to derive and design specific application scenarios in a systematic and reusable way.

4. Methodology for Developing Environments for Analysis of Molecular Docking Results

The methodology complements the framework described in the previous section by explaining how this

framework can be used during development. It clearly states the required roles (Life Scientist, Bioinformatician, Modeller, Software Developer, IT Infrastructure Administrator) and the specific deliverables (Diagram, Textual Description, Formal Description, Coding) for which these roles need to collaborate. The methodology is based on the seven principles identified by Cockburn [50]. A version of Cockburn's *role-deliverable-milestone* diagram has been created to represent the methodology (Figure 5). In this type of diagram each role is drawn in a separate box along with a list of all deliverables that are related to that role. On top of the diagram all the milestones are listed. This provides a concise way for a person who has been given a role to view which milestone of which deliverable (s)he is required to work on. Reading the diagram vertically shows which roles need to work together to complete a milestone.

Figure 5 illustrates that the Modeller, Life Scientist and Bioinformatician shall collaborate when creating the diagram and textual description of the scenario. Furthermore, the Modeller must collaborate with the Bioinformatician and the Software Developer when creating the formal description. Key components of this diagram, extensions to Cockburn's original diagram, are the dotted lines which show that the process is agile. This extension has been included to emphasise the need for a methodology that enables refinement and focuses on feedback from different roles. For instance, in the section where the Life Scientist works on the textual description (milestone M4 - M5), the dotted line shows that (s)he may revisit and alter the diagram if necessary. A key feature of this diagram is that it shows what role is needed for each deliverable and at which stage of the project. For instance, it shows that the Life Scientist is not required to produce the Formal Description. It also shows that during the coding, the Bioinformatician and Modeller may need to review the Formal Description. Finally, it shows that during the coding, the Modeller should work with the Software Developer to achieve milestone M11, while the Life Scientist and Bioinformatician should work with the Software Developer to achieve milestone M13. The asterisk (*) indicates that a similar but more detailed lower-level diagram of the coding has been developed (not presented in this paper but available in [49]).

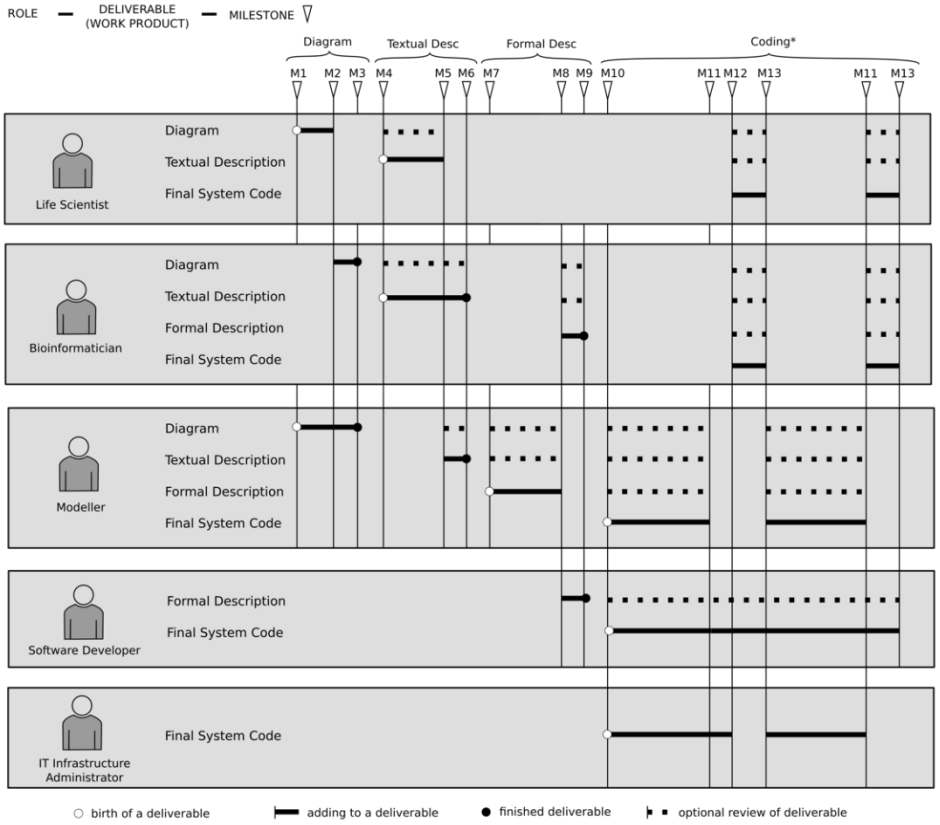

Figure 5 – Role-deliverable-milestone diagram of the proposed methodology. Roles on left, deliverables for each role in grey, milestones shown at top (M1: Diagram start; M2: Diagram ready for review; M3: Diagram ready; M4: Textual description start; M5: Textual description ready for review; M6: Textual description ready; M7: Formal description start; M8: Formal description ready for review; M9: Formal description ready; M10: Software development start; M11: Software developed; M12: Software installed/deployed; M13: Software tested).

Besides the above diagram, the methodology also provides three *techniques*, based on the framework presented in Section 3. They can be used iteratively by the software development team during milestones M1-M9 because the

diagram, textual, and formal descriptions are used in the techniques.

Technique 1. The basic diagram of the entire scenario can be used to determine whether the scenario fits the framework.

Technique 2. The abstract (diagrammatic, textual, and formal) description of an element can be used to determine whether there is a similar element, already implemented using the framework, that can be reused.

Technique 3. The abstract (diagrammatic, textual, and formal) description of a candidate tool can be used to determine whether this tool can be applied as an element in a scenario, by comparing it to the abstract description of a generic element type. This technique can be used if, using the framework, there is no already implemented element that can be reused, but there is an existing tool that has been implemented outside the framework (here called: candidate tool).

The application of these techniques will be illustrated in Section 5 when demonstrating and evaluating the utilisation and effectiveness of the framework and the methodology.

5. Evaluation of the framework and methodology using selected scenarios

The evaluation of the framework and the methodology was divided into two phases. In the first phase, it was proven that the framework is generic enough to support at least the five identified scenarios and the 14 related solutions investigated in Section 2. In order to verify this, a basic diagram, similar to the ones presented in Figure 6 in case of Scenario 1 and Figure 10 in case of Scenario 2, has been derived from the basic diagram of the generic framework (Figure 1) for each of the 19 (5 scenarios + 14 related work) test cases. As these diagrams are based on the basic diagram of the generic framework and utilise some or all of its element types, it is reasonable to say that all five scenarios and 14 solutions identified from the literature fit the framework. It is also noted that, in this step, Technique 1 from the methodology was applied to identify whether the scenarios fit the framework.

In the second phase, in order to demonstrate how the framework and the methodology support implementing molecular docking science gateways, we developed an implementation of Scenarios 1 and 2 following the methodology and utilising its three techniques. In the remaining part of this section these implementations are detailed highlighting the utilisation of the framework and the methodology. Extracts of the diagrammatic descriptions are presented here, while the complete version of these descriptions together with the developed code is available at [49].

5.1 Design and Implementation of Scenario 1

In Scenario 1 the intended system is required to assist the scientist by searching for *good* previous docking results that have used a receptor *similar* to the currently used one. Based on this similarity the system shall suggest a new receptor-ligand pair that would be an interesting candidate for a future docking run. The docking results and receptor data stored in the MDRR could have been produced by the same scientist or other scientists in the past. Scientists may have used different MDEs to insert docking results and receptor data in the MDRR.

As a first step, the basic diagram of the scenario (Figure 6) has been developed by replacing the building blocks of the generic framework (Figure 1) with functional elements supporting this particular scenario. At this step, Technique 1 of the methodology has been applied to assess which components of the generic framework are required and what their desired functionality is to implement the scenario. If such diagram can be derived, then the scenario fits the framework. Please note that at this stage we are not concerned about the actual implementation of the elements only about their desired functionality.

Scenario 1 requires an MDE that supports docking or large-scale VS experiments. The MDRR must store at least the docking results and data on the receptor used. Three ATs can be utilised in this scenario. An AT to calculate similarities between the structure of receptors known as structural alignment (AT1), an AT to assess whether the structural alignment result shows that the two receptors are *similar* (AT2), and an AT to assess a docking result and categorise it as *good* (AT3). Finally, a DM is needed to summarise the analysis and suggest which receptor-ligand pair to dock next. When compared to the basic diagram of the generic framework on Figure 1, we can state that an ADS is not required in this scenario. However, all other components of the framework have been utilised and mapped to desired and appropriate functional elements. The flow of data is shown in Figure 6.

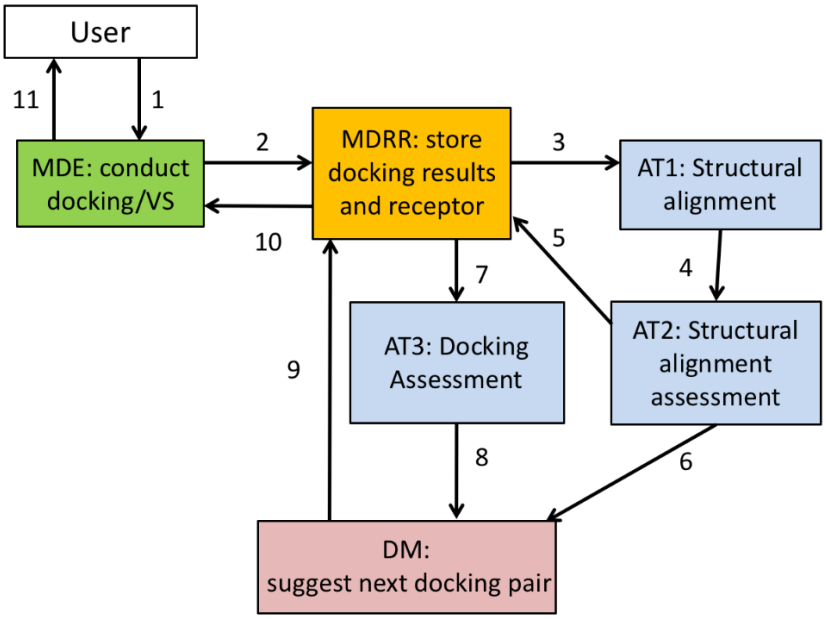

Figure 6 – Basic diagram of Scenario 1. User conducts docking (1) and results may be sent to the MDRR (2). The MDRR sends receptor pairs to AT1 (3) which calculates receptor similarities and sends results to AT2 for assessment (4). AT2 sends feedback to the MDRR (5) and the DM (6). Past docking results of similar receptors are assessed by AT3 (7) and only *good* results are sent to the DM (8). The DM combines results from ATs, suggests a receptor-ligand pair to dock in the future, and sends it for storage in the MDRR (9), which sends it to the MDE (10) and the user (11).

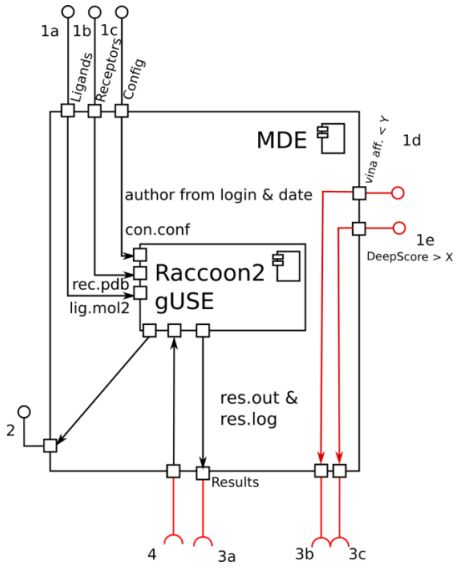

Figure 7 – Extract of the detailed diagram of Scenario 1. Based on the type of input/output interface 1 is broken down into 1a-e, while interface 3 into 3a-c. Data from 1a-c is used as input to the subcomponent responsible for VS, Raccoon2 gUSE. Data from 1d (AutoDock Vina threshold), and 1e (DeepAlign threshold) is forwarded.

Based on the detailed diagram of the generic framework and the basic diagram of the scenario, a detailed diagram of each scenario can be derived, followed by textual and formal descriptions of these scenarios. At this stage, the functional elements of the basic diagram are replaced by either existing already implemented components or by custom components that need to be implemented. This process is conducted iteratively, utilising Techniques 2 or 3 of the methodology in order to determine whether an already implemented tool can be reused. Technique 2 requires a library of abstract descriptions of already implemented elements, which did not exist prior to the design and implementation of Scenario 1. Section 5.2 shows how this technique can be used for Scenario 2, while the following paragraphs show the use of Technique 3.

In Scenario 1, any existing docking tool can be used as MDE, if it fits the description of element type MDE. One candidate can be the cloud-enabled version of Raccoon2, described in detail in our previous paper [51]. In this solution, WS-PGRADE/gUSE was integrated with Raccoon2 to support large-scale experiments on heterogeneous cloud computing resources. To determine whether the extended version of Raccoon2 can be used, following Technique 3, a diagrammatic (Figure 7), textual, and formal (Figure 8) description of the tool and its interfaces shall be created. The detailed diagram of Raccoon2 in Figure 7 can be derived from the detailed diagram of the generic element type MDE (top left in Figure 2). The core computation that is required (conduct docking/VS simulations) is achieved by the Raccoon2 solution. It is evident that the defined types of interfaces already exist as part of this solution (marked in black in Figure 7), but there is a need to further extend Raccoon2 to include additional interfaces, for example to facilitate communication with the MDRR (red in Figure 7).

The textual description can be derived from the textual description of the generic interfaces shown in Section 3 and it includes the interfaces presented in Figure 7. Examples of the additional (red) interfaces required include:

3a-c. Raccoon2 → MDRR, Raccoon2 needs to provide an interface to the docking results which may be sent to the MDRR, and the receptor and user-provided thresholds which shall be sent to the MDRR.

4. MDRR → Raccoon2, Raccoon2 requires an interface to receive suggested ligand for next docking from MDRR.

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

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

Docking_AutoDockVina

$$ligand? : LIGAND$$
$$receptor? : RECEPTOR$$
$$config? : CONFIG$$
$$result! : RESULT$$
$$config? \neq \varnothing \land result! = dockingWithConfig(ligand?, receptor?, config?)$$

MolecularDockingEnvironment_Raccoon2

$$ligands? : LIGANDS$$
$$receptors? : RECEPTORS$$
$$config? : CONFIG$$
$$results! : RESULTS$$
$$date! : DATE$$
$$\exists\, ligand? : ligands?;\ receptor? : receptors?;\ result! : results! \bullet Docking\_AutoDockVina$$

Figure 8 – Extract of the formal description of Scenario 1. A specific MDE which uses Raccoon2 and the function *dockingWithConfig()* whose main formula reads: for each non-empty ligand *l* and receptor r received, there exists a non-empty config *c* and docking result *res*, such that *dockingWithConfig(l, r, c) = res*.

Similarly, a formal description can be used to deduce whether Raccoon2 can be applied as an element in Scenario 1. A manual comparison of the formal descriptions which are written using Z notation is needed. If the formal description of the candidate tool can be derived from the formal description of the generic element type, then this candidate tool can be applied as an element. Otherwise, it cannot. Figure 8 shows that the description of Raccoon2 using Z notation is derived from the description of the generic element type MDE using Z notation (Figures 3 and 4). The definition of *dockingWithConfig* is used because docking with Raccoon2 always requires a config file. Analogously to the *Docking* and *MolecularDockingEnvironment* schemas of Figures 3 and 4, the schema *Docking_AutoDockVina* can be included within the schema *MolecularDockingEnvironment_Raccoon2*.

Using Technique 3, similarly to the process detailed above in case of the MDE, we have determined that the existing structural alignment tool DeepAlign can be used as the element AT1. We have also concluded that there is a need to develop custom-made tools to be used as AT2, AT3, DM, and MDDR. Additionally, as specified in the methodology, life scientists from the University of Westminster (UoW) took part in the development process, e.g. when deciding the molecular properties stored in the MDRR.

Based on the design considerations illustrated above, we can implement Scenario 1. The diagrammatic, formal, and textual abstract description of the elements, as well as the use of the three techniques, show which specific tools

can be used in an implementation. The framework and methodology are independent of the programming language or database technology applied. In our example, Python, and the micro web-framework Bottle [52] were used to implement basic RESTful APIs for every element. The Python module PyBel [53] was used to calculate molecular properties of ligands and receptors and they were stored in a MongoDB database as part of the MDRR. One of the reasons for choosing MongoDB was its schema-less design which enables using a single collection to store different formats of data from the MDE or the ATs.

Figure 9 shows the architecture of this implementation of Scenario 1. The solid lines represent the communication between servers through HTTP, while the dashed lines represent communication between different objects within one server (e.g. AT1 --> AT2). In this implementation, the MDRR receives the output from the MDE as a HTTP request (1) and stores it. Then, all unique receptors in the database are sent to the AT:DeepAlign along with the target receptor that was used in the MDE and the user-input DeepAlign threshold value (2). DeepAlign is run to find similarities between the target receptor and each additional receptor it received. The results are sent to AT:AssessDeepAlign (3) which selects the *similar* receptors by comparing the results of DeepAlign to the user-input threshold and returns it to the DM (4) which forwards them to the MDRR (7).

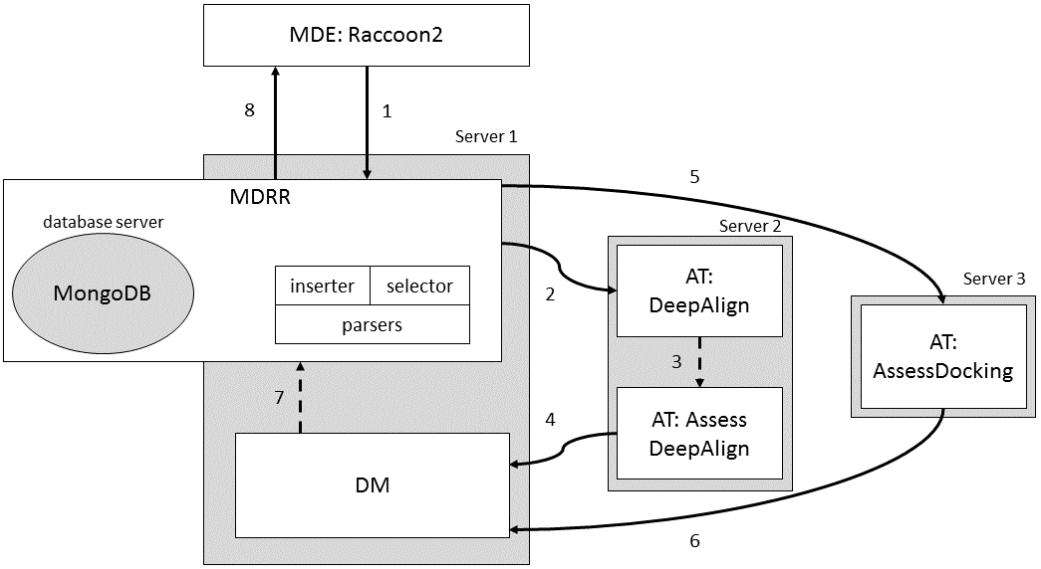

Figure 9 – Architecture of implementation of Scenario 1. Grey rectangles signify HTTP servers; white blocks signify elements. In this implementation, the MDRR contains a MongoDB database which is deployed on a database server, while AT: DeepAlign and AT: AssessDeepAlign are deployed on the same server.

In the next step, the MDRR sends previous docking results that have used a *similar* receptor to the AT:AssessDocking, along with the user-input AutoDock Vina affinity threshold (5). The AT:AssessDocking selects a docking result with affinity lower than this threshold, labels it a *good* docking result, and returns it to the DM (6). The DM combines these two lists and returns a list, sorted firstly based on the DeepAlign result value then on the affinity, to the MDRR (7) which presents the results to the MDE (8).

5.2 Design and Implementation of Scenario 2

The design of Scenario 1 illustrated how Techniques 1 and 3 can be applied. To illustrate how Technique 2 works, the design and implementation of Scenario 2 is presented here. In Scenario 2, the user runs a VS experiment, then ligands of *good* docking results are filtered based on a property available in an external ADS. The result is a sub-list of ligands that are more likely to produce useful laboratory results.

A basic diagram of Scenario 2 (Figure 10) can be derived from the basic diagram of the framework, similarly to the process explained in Section 5.1. Thus, we can conclude that Scenario 2 fits the framework (Technique 1). To provide a more precise analysis, the detailed diagram for each element and their interfaces can be determined. A list of the interfaces and a formal description were generated iteratively to confirm that the selected existing elements can be used in Scenario 2 prior to starting the coding step. The design of Scenario 2 used both Technique 2 and Technique 3 but this section only focuses on segments that are different from the design of Scenario 1, and comments on the added value of implementing Scenario 2.

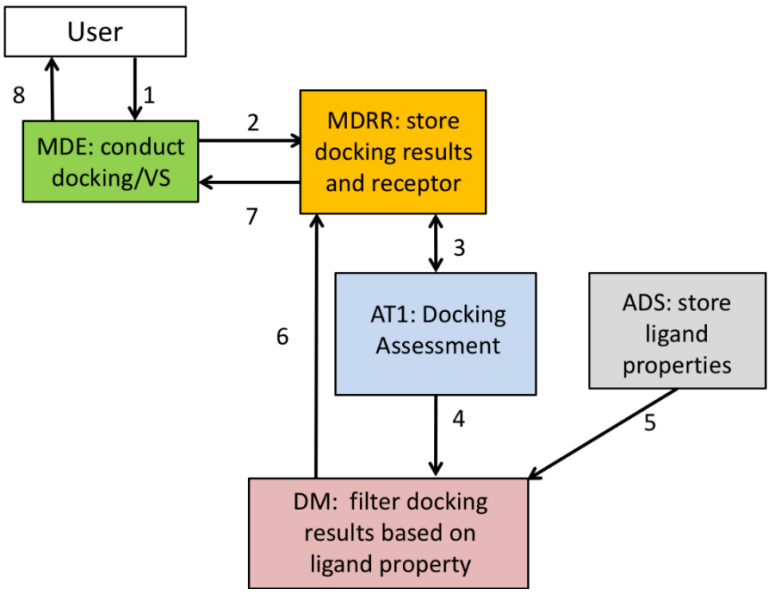

Figure 10 – Basic diagram of Scenario 2. User conducts docking (1) and results are sent to the MDRR (2). The MDRR sends docking results to AT1 which assesses whether the docking is *good* and returns feedback to the MDRR (3) and the DM (4). The DM combines this with data about ligand properties from the ADS (5), sends a decision to the MDRR (6) which stores it and sends it to the MDE (7), which visualises it for the user (8).

The MDE from Scenario 1 can be reused with the only difference that instead of a DeepAlign threshold, the user now inputs a molecular property name and threshold. When drawing the detailed diagram of the needed MDE, it becomes evident that most of the interfaces are the same, and the core computation (the docking) is the same as in the detailed diagram of Raccoon2 used for Scenario 1. The conclusion is that the abstract description of Raccoon2 is similar enough for it to be used as an MDE in Scenario 2. The fact that the element can be reused is determined by searching a library of abstract descriptions of already implemented elements (Technique 2). This library contains only one MDE, but the same method of drawing the detailed diagram and comparing the interfaces and the core computation, can be used to search a larger library. When writing the formal description of the needed MDE we noticed that the Z notation is practically the same as the only Z notation of an MDE currently in the library. Manually comparing the formal description of the required element and the element in the library led to the decision that the MDE in the library can be reused in Scenario 2.

Technique 2 can also be used to determine which tool can be applied as AT to assess docking in Scenario 2. There is a library of three already implemented ATs from Scenario 1. Their diagrammatic abstract descriptions are shown in Figure 11B, C, and D. When the AT required in Scenario 2 (Figure 11A) is compared to AT:DeepAlign (Figure 11B), there is a clear difference in the interfaces. AT:DeepAlign needs to send the results and a user-provided threshold to another AT for assessment. Whereas the required AT assesses docking results and sends them to an MDRR for storage, and a DM for summarising. When compared to AT:AssessDeepAlign (Figure 11C), there are more similarities in the interfaces. The difference is that AT:AssessDeepAlign requires input from another AT, whereas the required AT needs input from an MDRR. However, there is a big difference in the core computation. While AT:AssessDeepAlign filters structural alignment results, the required AT needs to filter docking results. Finally, when comparing the required AT with AT:AssessDocking (Figure 11D), it is clear that the interfaces are the same (requires input from MDRR, provides results to MDRR and DM), and the core computation is the same (assess docking results). Therefore, AT:AssessDocking can be reused as a building block of Scenario 2.

Additionally, Scenario 2 demonstrates how a new element type can be introduced. As it was noted before, in Scenario 1 there was no ADS. Following Technique 3, we determined that PubChem [54], a repository that contains data about various properties of chemical compounds, can be used as ADS in Scenario 2. Data stored in PubChem can be accessed programmatically through the Power User Gateway (PUG) REST API. Thus, this existing interface can be used to obtain the needed value for the molecular property of the ligands, and PubChem can be used as an ADS. Finally, it can be noted that the DM is an element that is typically specific to each scenario, due to the inherent difference in the core computation.

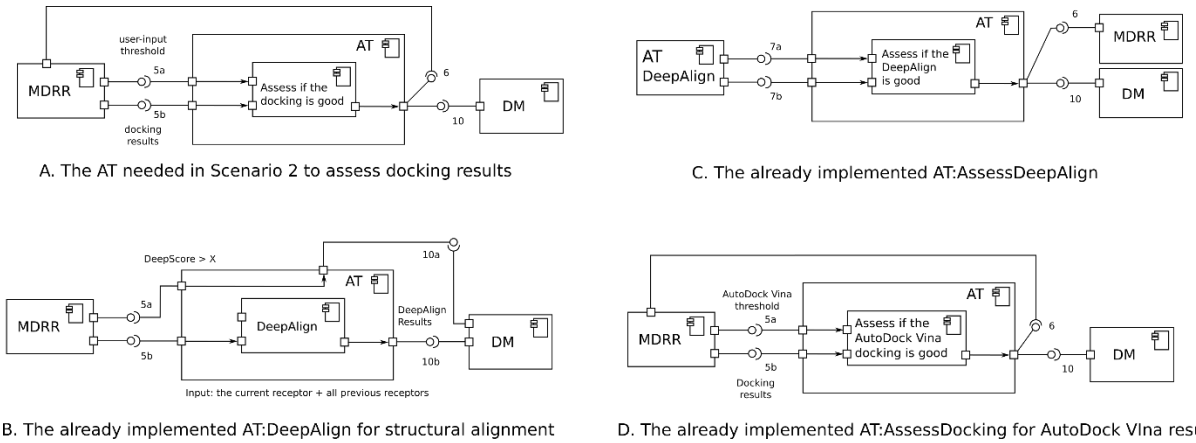

Figure 11 – Diagrammatic abstract descriptions of the required AT (A) and the three elements already present in the library of abstract descriptions (B, C, D). This diagram is used to determine whether any already implemented elements can be reused – note the similarities between interfaces and core computation between A and D.

This section illustrated the use of Techniques 1, 2, and 3. Technique 1 uses the basic diagram, while Techniques 2 and 3 use the diagrammatic, textual, and formal description of elements. In the design of Scenario 2, we only show the use of the diagrammatic description, but the textual and formal descriptions can be used analogously.

Based on the above design of Scenario 2, the code can be implemented (source code available in [49]). In the implementation of Scenario 2, similarly to the implementation of Scenario 1, all components are accessible via a RESTful API developed using the Bottle web framework (Figure 12). The docking results are sent from the MDE to the MDRR (1), which forwards them to the AT:AssessDocking (2). Filtered *good* docking results are returned to the DM (3) which forwards them to the MDRR (6). Then a ligand identifier, known as SMILES code, is selected for each ligand that was part of docking results that passed the filter. The MDRR sends these ligand identifiers to the ADS (4) which checks if a user-input ligand property is above or below a threshold and returns the result to the DM (5). Finally, the DM summarises the results and sends them to the MDRR (6) which sends them to the MDE for visualisation (7).

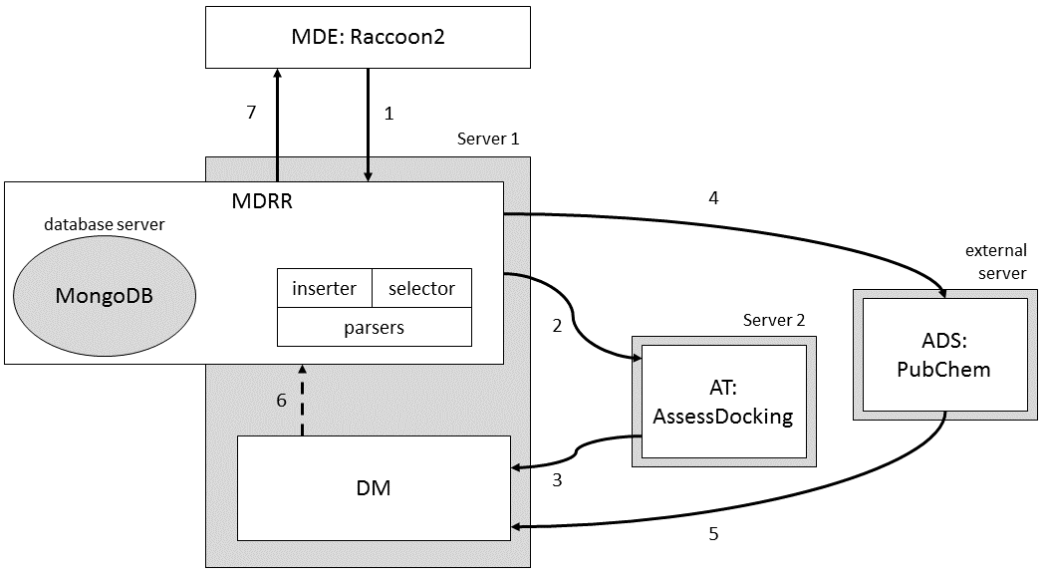

Figure 12 – Architecture of implementation of Scenario 2. Grey rectangles signify HTTP servers; white blocks signify elements. In this implementation, the MDRR contains a MongoDB database which is deployed on a database server, the AT is deployed on a separate server, while the ADS is deployed on an existing external server.

One of the aims of implementing Scenario 2 was to show how an element that was created in an implementation that has utilised the framework and methodology, can be reused. Evidence for this are the elements MDE, the AT:AssessDocking, and to some extent the MDRR, which are reused with nearly no alterations. The minor addition

was the change of the user-input threshold values required in the MDE, while the AT:AssessDocking was reused without any code changes. Another aim was to illustrate how an additional element such as ADS can be included (since an ADS was not present in Scenario 1). It is demonstrated in Scenario 2 that the external database PubChem and its interfaces can be applied as ADS. Therefore, the MDRR would need to be slightly altered in order to facilitate the communication with PubChem.

6. Usability of implementations that utilise the framework

Using the framework and methodology provides benefits for software developers by allowing them to determine the specific tools needed and reusing existing elements in an efficient way, as shown in Section 5. This is an alternative to acquiring the domain knowledge to make this determination. Nevertheless, evidence was still required to demonstrate that the designed and implemented solutions are useful for the end-users, the scientists. In order to prove that following such a methodical approach does not produce cumbersome and less usable systems, we have conducted several usability tests. This section aims to describe the usability of solutions implemented with the framework, but it does not aim to present a quantification of the benefits for the end-user. Completing a scenario using currently available tools directly ("without" using the framework) was compared to completing the scenario "with" the implementation that was based on the framework. The results for Scenario 1, based on the implementation described in Section 5.1, are presented in this paper. The focus of the usability tests was the additional analysis provided as a result of the scenarios, and not the results of the docking simulations themselves. The docking experiments used the ribokinase of a protozoan parasite *Trichomonas vaginalis* (TV) as a receptor, an adequate config file, and a set of 10 ligands. The objective was to run this docking experiment and then suggest a receptor-ligand pair that should be used in the next molecular docking, based on receptor similarity and previous docking results. The usability tests demonstrated that the experiment can be successfully completed with the implemented solution and that it provides significant benefits to the scientists when compared to manual execution.

As a preparation for the tests, the MDRR needed to be filled with relevant previous docking results. It was required that the previous docking results include at least one receptor that is *similar* to our target receptor and has a *good* docking stored in the MDRR. Therefore, a large number of docking simulations were conducted using the extended Raccoon2. The private OpenStack-based cloud at the University of Westminster (UoW Cloud) was used as the DCI to run the docking simulations. When choosing which receptors to dock in order to fill the MDRR, several *a priori* similar receptors to the TV ribokinase were chosen. Structures of 7 receptors of same type (ribokinase) from different species were downloaded from the wwPDB [55]. To ensure that there is structural alignment, DeepAlign was run between the TV ribokinase and each of the 7 ribokinases resulting in a high DeepScore value (between 975.47 and 1491.79). Further 63 receptors were chosen from the RCSB "Molecule of the Month" series [56]. Therefore, a total of 70 receptors were used when prefilling the MDRR, 7 of them (10%) *a priori* similar to the TV ribokinase, and 63 (90%) other receptors. Each receptor required its own configuration file which was created within the GUI of Raccoon2. Care was taken for the cuboid of the config file to cover a part of the receptor, but further analysis for biological relevance was not conducted. Finally, a large number of ligands was needed. A total of 2,376 molecules approved as drugs somewhere in the world was obtained from ZINC [57] and made up our set of ligands.

These input files were used to fill the MDRR with 166,320 (70 × 2,376) docking results. For each receptor, between 3 and 6 jobs were run on the UoW Cloud, resulting in 393 jobs with mean execution time of 2h 23min 23s.

Following these preparations, Scenario 1 was completed first "without" using the implementation of Section 5.1 but manually feeding results into the appropriate tools. This process was then compared to executing the same scenario "with" the implementation.

One observation is that the usability tests "with" the implementation contain fewer manual steps than the usability tests "without" the implementation ("with" required 3, while "without" required 6 manual steps). This is because most of the steps are automatised. There is no need to locate files on the file system, or manually read through web pages. The decrease of manual steps is a major benefit in terms of usability.

Another benefit is the decrease of complexity of the manual steps. Generally, the user needs to prepare the docking or VS in Raccoon2, enter the required user-provided inputs, and wait for the result of the scenario. When completing the scenario "without" the implementation, the user would additionally need to complete steps that may be considered complex for a biomedical scientist, such as writing and running a small script that would rename docking result files to contain the DeepAlign result value in their name.

Furthermore, running the scenarios "without" the implementation sometimes requires simple, but repetitive manual steps such as uploading the structures of receptors and reading the results of the structural alignment multiple times. Although not complex, these repetitive manual steps are very error-prone and time-consuming. Using the implementations does not have these problems.

7. Conclusion and Future Work

This paper presented a generic framework and a corresponding methodology to implement complex science-gateway-based environments for the execution of molecular docking experiments extended with the intelligent analysis and utilisation of docking results. The framework incorporates a diagrammatic, textual, and formal description enabling a modular design and the replacement and reuse of components. The methodology involves multiple stakeholders and requires their collaboration in an agile manner. It provides three techniques that can be used to determine the specific elements that can be used in a scenario. Following the framework and methodology, the implementations of two scenarios were developed and presented.

The framework and methodology enable the systematic and efficient reuse of existing elements and therefore formalise and potentially speed-up the development process. Additionally, users of the implementations, the biomedical scientists, will benefit from implementing these types of scenarios using the framework and methodology which produces usable systems providing relevant results. The three techniques that are part of the methodology require manual comparison between the diagrams, text, and formal descriptions. As future work, we propose investigating whether a software tool can be developed to conduct this comparison automatically.

Although the framework and methodology have been created for systems that store and analyse docking results, a similar approach can be used for other domains. The compatibility and effectiveness of this approach in related domains in bioinformatics such as the more computationally demanding molecular dynamics simulations could also be investigated.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007–2013) under grant agreement No.608886(CloudSME) and from the H2020 Programme under Grant Agreement No.731574 (COLA). The authors would also like to acknowledge funding from the University of Westminster Research Studentship 2014.

References

[1] J. C. Foreman, T. Johansen and A. J. Gibb, *Textbook of receptor pharmacology*. CRC press, 2010.

[2] D. Temelkovski, T. Kiss, and G. Terstyanszky, A Generic Framework and Methodology for Implementing Science Gateways for Analysing Molecular Docking Results. Proc. of 10th IWSG 2018, Edinburgh, UK, 13-15 Jun, 2018, CEUR-WS.org, online http://ceur-ws.org/Vol-2357/paper14.pdf.

[3] I. D. Kuntz, J. M. Blaney, S. J. Oatley, R. Langridge, and T. E. Ferrin, A geometric approach to macromolecule-ligand interactions, J. Mol. Bio., vol. 161, no. 2, pp. 269-288, 1982.

[4] W. J. Allen, T. E. Balius, S. Mukherjee, S. R. Brozell, D. T. Moustakas, P. T. Lang,D. A. Case, I. D. Kuntz, and R. C. Rizzo, DOCK 6: Impact of new features and current docking performance, J. Comp. Chem., vol. 36, no. 15,pp. 1132-1156, 2015.

[5] Z. Vincent and D. Antoine, Click2Drug: Directory of in silico drug design tools, Sep 2017. Available at: http://www.click2drug.org/index.html#Screening [Accessed 21 Feb 2020]

[6] G. Jones, P. Willett, R. C. Glen, A. R. Leach, and R. Taylor, Development and validation of a genetic algorithm for flexible docking, J. Mol. Bio., vol. 267, no. 3, pp. 727-748, 1997

[7] B. Kramer, M. Rarey, and T. Lengauer, Evaluation of the FlexX incremental construction algorithm for protein-ligand docking, Proteins, vol. 37, no. 2, pp. 228-241, 1999.

[8] G. M. Morris, R. Huey, W. Lindstrom, M. F. Sanner, R. K. Belew, D. S. Goodsell, and A. J. Olson, AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility, J, Comp. Chem., vol. 30, no. 16, pp. 2785-2791, 2009.

[9] S. F. Sousa, P. A. Fernandes, and M. J. Ramos, Protein-ligand docking: Current status and future challenges, Proteins, vol. 65, pp. 15-26, Jul 2006.

[10] O. Trott and A. J. Olson, AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading, J. Comp. Chem., pp. 455-461, 2009.

[11] M. W. Chang, C. Ayeni, S. Breuer, and B. E. Torbett, Virtual screening for HIV protease inhibitors: A comparison of AutoDock 4 and Vina, PLoS ONE, vol. 5, no. 8, p. e11955, 2010.

[12] S. Forli, R. Huey, M. E. Pique, M. F. Sanner, D. S. Goodsell, and A. J. Olson, Computational protein-ligand docking and virtual drug screening with the AutoDock suite, Nature Protocols, vol. 11, no. 5, p. 905, 2016.

[13] H. Hasegawa and L. Holm, Advances and pitfalls of protein structural alignment, Current Opinion in Structural Biology, vol. 19, no. 3, pp. 341-348, 2009.
[14] E. C. Meng, Online structure alignment resources, Apr 2005. Available at: http://www.rbvi.ucsf.edu/home/meng/grpmt/structalign.html [Accessed 21 Feb 2020]
[15] E. Martz, W. Decatur, and M. Wiederstein, Structural alignment tools, Oct 2016. Available at: http://proteopedia.org/wiki/index.php/Structural_alignment_tools [Accessed 21 Feb 2020].
[16] L. Holm and C. Sander, Protein structure comparison by alignment of distance matrices, J. Mol. Bio., vol. 233, no. 1, pp. 123-138, 1993.
[17] I. N. Shindyalov and P. E. Bourne, Protein structure alignment by incremental combinatorial extension (CE) of the optimal path, Protein Engineering, vol. 11, no. 9, pp. 739-747, 1998.
[18] S. Wang, J. Ma, J. Peng, and J. Xu, Protein structure alignment beyond spatial proximity, Scientific Reports, vol. 3, p. 1448, 2013.
[19] Object Management Group, Unified Modeling Language Version 2.5.1. Available at: https://www.omg.org/spec/UML/2.5.1 [Accessed 21 Feb 2020]
[20] Object Management Group, The OMG Systems Modeling Language Version 1.6. Available at: https://www.omg.org/spec/SysML/1.6/ [Accessed 21 Feb 2020]
[21] P. Cuenot, et al., The EAST-ADL Architecture Description Language for Automotive Embedded Software, Chapter 11 in *Model-Based Engineering of Embedded Real-Time Systems*, Ed. Holger Geise et al., pp. 297-388, 2010.
[22] J.S. Fitzgerald, P.G. Larsen, and M. Verhoef, Vienna development method. Wiley Encyclopedia of Computer Science and Engineering, pp.1-11, 2007.
[23] D. Jackson, Alloy: a lightweight object modelling notation. ACM Transactions on Software Engineering and Methodology (TOSEM), 11(2), pp.256-290, 2002.
[24] J. M. Spivey, The Z notation: A reference manual, tech. rep., Oxford: Oriel College, 1998. Available at: https://www.cse.buffalo.edu/LRG/CSE705/Papers/Z-Ref-Manual.pdf [Accessed 21 Feb 2020]
[25] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, Scientific workflow management and the Kepler system, Concurrency and Computation: Practice and Experience, vol. 18, no. 10, pp. 1039-1065, 2006.
[26] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, Nextflow enables reproducible computational workflows, Nature biotechnology, vol. 35, no. 4, p. 316, 2017.
[27] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, et al., The Taverna workflow suite: Designing and executing workflows of web services on the desktop, web or in the cloud, Nucleic Acids Research, vol. 41, no. W1, pp. W557-W561, 2013.
[28] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, and I. Marton, WS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities, Journal of Grid Computing, vol. 10, no. 4, pp. 601-630, 2012.
[29] L. Xie, T. Evangelidis, L. Xie, and P. E. Bourne, Drug discovery using chemical systems biology: Weak inhibition of multiple kinases may contribute to the anti-cancer effect of nelfinavir, PLoS Comput. Biology, vol. 7, no. 4, p. e1002037, 2011.
[30] X. Zhang, S. E. Wong, and F. C. Lightstone, Toward fully automated high performance computing drug discovery: A massively parallel virtual screening pipeline for docking and molecular mechanics/generalized Born surface area rescoring to improve enrichment, J. Chem. Inf. Model., vol 54, no. 1, pp. 324-337, 2014.
[31] X. Jiang, K. Kumar, X. Hu, A. Wallqvist, and J. Reifman, DOVIS 2.0: An efficient and easy to use parallel virtual screening tool based on AutoDock 4.0, Chemistry Central Journal, vol. 2, no. 1, p. 18, 2008.
[32] P. D'Ursi, F. Chiappori, I. Merelli, P. Cozzi, E. Rovida, and L. Milanesi, Virtual screening pipeline and ligand modelling for H5N1 neuraminidase, Biochem. and Biophys. Res. Comm., vol. 383, no. 4, pp. 445-449, 2009.
[33] G. Van Zundert, J. Rodrigues, M. Trellet, C. Schmitz, P. Kastritis, E. Karaca, A. Melquiond, M. van Dijk, S. De Vries, and A. Bonvin, The HADDOCK2. 2 web server: user-friendly integrative modeling of biomolecular complexes. J. Mol. Bio., vol. 428 no.4, pp.720-725, 2016.
[34] T. Kiss, P. Borsody, G. Terstyanszky, S. Winter, P. Greenwell, S. McEldowney, and H. Heindl, Large-scale virtual screening experiments on Windows Azure-based cloud resources, Concurrency and Computation: Practice and Experience, vol. 26, no. 10, pp. 1760-1770, 2014.
[35] E. Glaab, Building a virtual ligand screening pipeline using free software: A survey, Briefings in Bioinformatics, vol. 17, no. 2, pp. 352-366, 2015.

23 Feb, 2020

[36] Z. Farkas, P. Kacsuk, Á. Hajnal, Enabling workflow-oriented science gateways to access multi-cloud systems, J. Grid Computing vol. 14, no. 4, pp. 619-640, 2016.
[37] P. Kacsuk (ed.), *Science Gateways for Distributed Computing Infrastructures: Development Framework and Exploitation by Scientific User Communities*, Springer, 2014. pp. 301.
[38] P, Kacsuk, Z. Farkas, M. Kozlovszky, G. Herman, A. Balasko, K. Karoczkai, I. Marton, WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities, J. Grid Computing, vol. 10, no. 4, pp 601-630, 2012.
[39] Z. Farkas, P. Kacsuk, T. Kiss, P. Borsody, Á. Hajnal, Á. Balaskó, and K. Karóczkai, Autodock gateway for molecular docking simulations in cloud systems, Cloud Computing with E-science Applications, p. 300, 2015.
[40] T. Kiss, P. Greenwell, H. Heindl, G. Terstyanszky, and N. Weingarten, Parameter sweep workflows for modelling carbohydrate recognition, J. Grid Computing, vol. 8, no. 4, pp. 587-601, 2010.
[41] M. Jaghoori, A. J. Altena, B. Bleijlevens, S. Ramezani, J. L. Font, and S. D. Olabarriaga, A multi-infrastructure gateway for virtual drug screening, Concurrency and Computation: Practice and Experience, vol. 27, no. 16, pp. 4478-4490, 2015.
[42] J. Krüger, R. Grunzke, S. Gesing, S. Breuers, A. Brinkmann, L. de la Garza, O. Kohlbacher, M. Kruse, W. E. Nagel, L. Packschies, et al., The MoSGrid science gateway a complete solution for molecular simulations, J. Chem. Theory and Computation, vol. 10, no. 6, pp. 2232-2245, 2014.
[43] C.A. Goble, and D.C. De Roure, myExperiment: social networking for workflow-using e-scientists. In Proceedings of the 2nd workshop on Workflows in support of large-scale science (pp. 1-2). ACM, 2007.
[44] G. Terstyanszky, T. Kukla, T. Kiss, P. Kacsuk, A. Balasko, Z. Farkas, Enabling Scientific Workflow Sharing through Coarse-Grained Interoperability, Future Generation Computing Systems: The International Journal of Grid Computing and eScience, vol 37, pp. 46-59, 2014.
[45] A. Roy, B. Srinivasan, and J. Skolnick, PoLi: A virtual screening pipeline based on template pocket and ligand similarity, J. Chem. Inf. Model., vol. 55, no. 8, pp. 1757□1770, 2015.
[46] T. A. Wassenaar, M. Van Dijk, N. Loureiro-Ferreira, G. Van Der Schot, S. J. De Vries, C. Schmitz, J. Van Der Zwan, R. Boelens, A. Giachetti, L. Ferella, et al., WeNMR: Structural biology on the grid, J. Grid Computing, vol. 10, no. 4, pp. 743-767, 2012.
[47] E. Chia, M. S. Shamsir, Z. A. Hussein, and S. Z. M. Hashim, GridMACS portal: A grid web portal for molecular dynamics simulation using GROMACS, in Mathematical/Analytical Modelling and Computer Simulation (AMS), 2010 Fourth Asia International Conference on, pp. 507-512, IEEE, 2010.
[48] P. Kunszt, L. Blum, B. Hullár, E. Schmid, A. Srebniak, W. Wolski, B. Rinn, F.-J. Elmer, C. Ramakrishnan, A. Quandt, et al., iPortal: The swiss grid proteomics portal: Requirements and new features based on experience and usability considerations, Concurrency and Computation: Practice and Experience, vol. 27, no. 2, pp. 433-445, 2015.
[49] D. Temelkovski, Implementation of scenarios, source-code on GitHub. Available at https://github.com/damjanmk/mdrr-scenarios [Accessed: 21 Feb 2020]
[50] A. Cockburn, *Agile software development: The cooperative game*. Pearson Education, 2nd ed., 2006.
[51] D. Temelkovski, T. Kiss, G. Terstyanszky, and P. Greenwell, Extending molecular docking desktop applications with cloud computing support and analysis of results, Future Generation Computer Systems, vol. 97, pp. 814-824, 2019.
[52] M. Hellkamp, Bottle: Python Web Framework Bottle 0.13-dev documentation, Jan 2019. Available at https://bottlepy.org/docs/stable/ [Accessed: 21 Feb 2020]
[53] N. M. O'Boyle, C. Morley, and G. R. Hutchison, Pybel: A Python wrapper for the OpenBabel cheminformatics toolkit, Chemistry Central Journal, vol. 2, no. 1, p. 5, 2008.
[54] Kim, P. A. Thiessen, E. E. Bolton, J. Chen, G. Fu, A. Gindulyte, L. Han, J. He, S. He, B. A. Shoemaker, et al., PubChem Substance and Compound databases, Nucleic Acids Research, vol. 44, no. D1, pp. D1202-D1213, 2015.
[55] H. Berman, K. Henrick, and H. Nakamura, Announcing the worldwide protein data bank, Nature Structural and Molecular Biology, vol. 10, no. 12, p. 980, 2003.
[56] D. S. Goodsell, S. Dutta, C. Zardecki, M. Voigt, H. M. Berman, and S. K. Burley, The RCSB PDB "molecule of the month": Inspiring a molecular view of biology, PLoS Biology, vol. 13, no. 5, p. e1002140, 2015.
[57] J. J. Irwin and B. K. Shoichet, ZINC - a free database of commercially available compounds for virtual screening, J. Chem. Inf. Model., vol. 45, no. 1, pp. 177-182, 2005.

23 Feb, 2020