Deliverable D3.7

Funding Scheme: THEME [ICT-2007.8.0] [FET Open]

Paving the Way for Future Emerging DNA-based Technologies: Computer-Aided Design and Manufacturing of DNA libraries

Grant Agreement number: 265505

Project acronym: CADMAD

Deliverable number: 3.7

Deliverable name: Specifications of a compiler for a programmable microfluidic platform

Contractual Date[1] of Delivery to the CEC: M12 (initial responsibility of UEVE)
Actual Date of Delivery to the CEC: M24
Author(s)[2]: Uwe Tangen, John McCaskill
Participant(s)[3]: RUB
Work Package: WP3
Security[4]: Pub
Nature[5]: R
Version[6]: 1.0
Total number of pages: **13**

1 As specified in Annex I

2 i.e. name of the person(s) responsible for the preparation of the document

3 Short name of partner(s) responsible for the deliverable

4 The Technical Annex of the project provides a list of deliverables to be submitted, with the following classification level:
Pub - Public document; No restrictions on access; may be given freely to any interested party or published openly on the web, provided the author and source are mentioned and the content is not altered.
Rest - Restricted circulation list (including Commission Project Officer). This circulation list will be designated in agreement with the source project. May not be given to persons or bodies not listed.
Int - Internal circulation within project (and Commission Project Officer). The deliverable cannot be disclosed to any third party outside the project.

5 **R (Report):** the deliverables consists in a document reporting the results of interest.
P (Prototype): the deliverable is actually consisting in a physical prototype**,** whose location and functionalities are described in the submitted document (however, the actual deliverable must be available for inspection and/or audit in the indicated place)
D (Demonstrator): the deliverable is a software program, a device or a physical set-up aimed to demonstrate a concept and described in the submitted document (however, the actual deliverable must be available for inspection and/or audit in the indicated place)
O (**Other):** the deliverable described in the submitted document can not be classified as one of the above (e.g. specification, tools, tests, etc.)

6 Two digits separated by a dot:
The first digit is 0 for draft, 1 for project approved document, 2 or more for further revisions (e.g. in case of non acceptance by the Commission) requiring explicit approval by the project itself;
The second digit is a number indicating minor changes to the document not requiring an explicit approval by the project.

Abstract

The microfluidic compiler is a hardware-dependent compiler and as such intimately linked to the microfluidic hardware used. The compiler uses droplets as basic computation units. The compiler targets primarily a highly reproducible scenario of linear (unbranched) droplet chains where the processing consists of extraction, separation processing and injection of DNA rather than merge and split of the droplets themselves. This requires that droplets can always be identified, do not merge (unless specifically activated to do so) and do not vanish from the system, apart from being washed out into one of the *k* outputs. The compiler needs to know the content of each input-droplet and must be able to extrapolate the physical properties of each particular droplet in the system. Though certain general principles apply, the physical dimensions and other details of the hardware must be incorporated into the compiler software.

Some basic operations are used by the compiler to find an algorithm satisfying the given physico-chemical rules: extraction of material from a droplet, insertion of material into a droplet, fluid-dynamic starting and stopping a particular lane of droplets, electrical transport and separation of chemicals in the intervening gel-network and some global properties, like temperature or buffer conditions.

Keywords[7]**:**

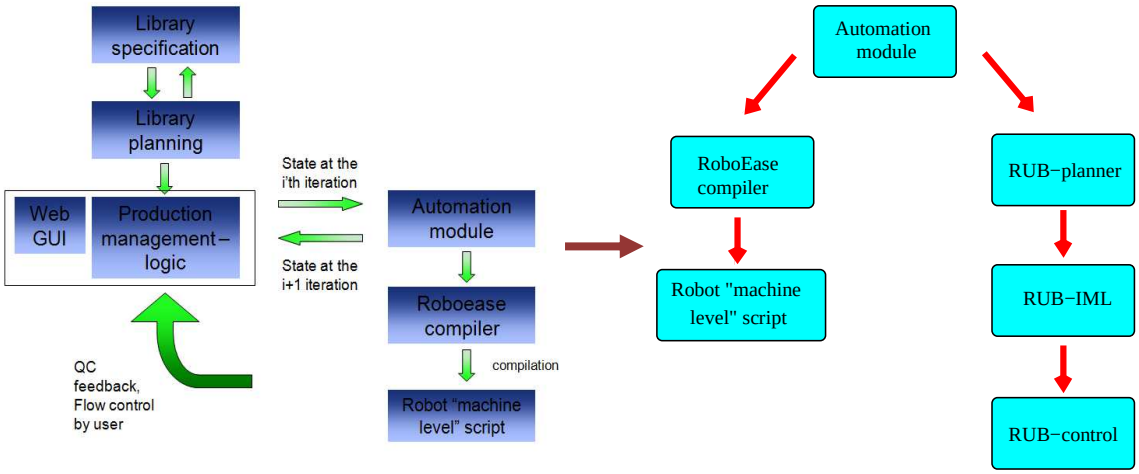
Microfluidics, DNA synthesis, compiler, on-chip separation, droplet.

[7] Keywords that would serve as search label for information retrieval

1. Aims and Objectives

The overall aim is to develop an open-source Microfluidic programming language and compiler. In contrast with the original proposal, this compiler is now being developed by RUB, and will target microfluidics integrated beyond the limit where microvalves can simply replace macroscopic liquid handling robot functions. Instead, the fundamental functional primitives will be electronic extraction from and injection into droplets and electronic on-chip gel separation, using microelectrode arrays. By using high-level programs, it will be possible to create large, self-directing experiments that are difficult to formulate at the hardware level using large robotic systems.

2. Results



The RUB-planner will be attached to the Automation Module of the CADMAD system.
The microfluidic compiler is part of the overall CADMAD software system and will be attached at the *Automation module* as RUB-planner (high-level frontend of the compiler) and RUB-IML an intermediate language specifying all the physical procedures and boundary conditions.

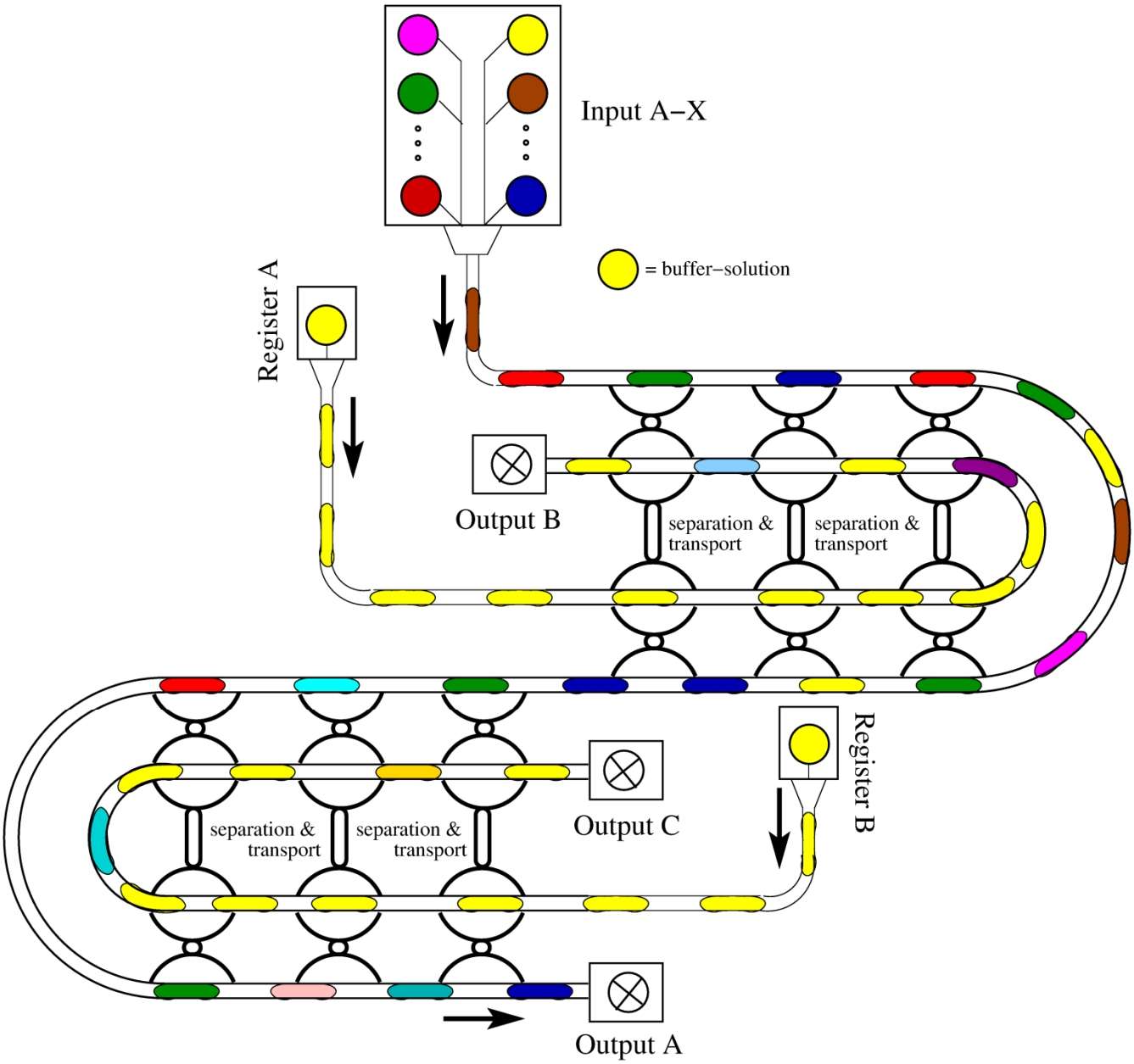
Topology of microfluidic computation device

The general hardware-topology is depicted in the following figure: chemicals available in reservoirs can be packed into droplets, either in pure form or in certain mixtures. These droplets must remain intact throughout the whole processing: They are and have to be discrete entities. The compiler, controlling the droplet-creation, has complete information about the contents of the input-droplets. These droplets are then fed into the input-lane and tracked throughout the processing. The compiler must keep track of the topological position of each droplet, it must also know all the traveling times of all droplets. If the traveling of droplets somehow deviates from the projected values the underlying control-hardware must assure a proper droplet movement. In case of failure, the corresponding error-flags must be set and the whole procedure driven into a safe error state. As such the compiler is a superset of a classical compiler and controller.

The principal idea of the manipulation of droplet content is to extract and insert material from as large as possible subset of *n* by *n* droplets, communicating via a dedicated network of electrical transportation or separation lanes. The routing topology is such that the content of a maximum number of droplets can be reinserted into upstream droplets to allow for an infinite iteration of chemical processes in the system. Because of losses and deterioration of chemicals in the droplets, amplification steps must be designed in into the chemical rules system. These amplification steps can be either isothermal sequence or temperature controlled PCR-like amplification procedures.
The topology shown in the following figure is a compromise between a fully random access-network and a microfluidic one-dimensional transport topology. Fully random access is not possible because of routing

constraints in the essentially planar system. Furthermore, this topology minimizes losses due to abrupt changes in tubings or valve operations. The design does not yet contain areas for droplet diversions, these are typically done with integrating Y-structures.

A design with two registers is shown in the following figure:



Input-set

The input is separated into two different sets: (i) the *set of available chemicals*, including the knowledge of all required physical and chemical properties and (ii) the *set of chemical rules* including timing, physical and chemical dependencies of and between the rules.

Chemicals

Chemicals can be either DNA sequences which have been created via the DNA Library Designer or other substances like enzymes or special buffers or salts.
The following parameters are required for each input chemical (apart from Name, all other parameters are provided with default-values if nothing is specified):

- Name
- Type
- Quantity
- Link to physical property database lookup
- Location at the input-device

The following properties may be required for each chemical (parameters are provided with default-values if nothing is specified):

- Detection method (lower and upper thresholds, concentration-intensity mapping, …)
- Transport coefficients (e.g. diffusion, electrophoresis)
- Concentration dependent contributions to viscosity and other solution properties
- Incompatibilities with other chemicals (list of names)
- Contamination propensity (nr. of washing steps required after moving along a location depending on the time of occupancy of a certain location)
- Temperature constraints
- Valid time range (min., max.) (e.g. with enzymes, the optimal time required for reaction)
- Required channel properties (e.g. special coating of channel walls, pH and salt-conditions etc.)

Rules (RUB-Planner)
The rules essentially map the algorithm into the system. There are chemical rules, which can either contain chemical names or types. As in SBML[i], reactions between chemicals can be encoded.
These rules are written in canonical form. Three types of chemical rules exist: rules which are active all the time and act in parallel, rules which are expressed conditionally, like in a classifier system and base operations which are changing the environment or add further control structures.

- Canonical form
  (using a more general form here than in DNApl is necessary because the compiler also has to deal with enzymes and other general chemical substances).

name: A + B ⟶ C + D

Without writing a symbol, emptiness is assumed (⟶ A means, A is created from nothing, which means insertion of a substance into a droplet, in the same manner A ⟶ means the extraction of a chemical from a droplet). The rule is only executed, in the compiler internal model, when the concentration of the source-chemicals is above a certain threshold. Of course, there can be *n* educts and *m* products in the equation, A+B→C+D is only an example .

- Conditional form, whether a condition is fulfilled depends on the measured concentration, if the according chemical was specified with a detection method. If no detection method is supplied a modeled assumption on the expected concentration is made.

name: A: execute rule X if concentration of A surpasses a certain threshold.
name: ! A: execute rule U if concentration of A falls below a certain threshold.
name: A . B: execute rule Y if concentration of A and concentration of B are above a certain threshold.

name: A + B: execute rule Z if concentration of A or concentration of B are above a certain threshold.

- Basic operations:

name: loop V, which executes rule V *k* times.
name: halt the execution of the system.
name: wait, wait for a specific amount of time, at a specified temperature.
name: mixture, a recipe to combine chemicals in a droplet. This is done only at droplet-chain-creation. A typical droplet volume is: 40um x 30um x 100um = 120pl. A volume of 0.3ul is equivalent to about 2300 droplets.
name: merge, *n* droplets are merged into a larger droplet.
name: select, select a droplet from *k* droplets.
name: sequence, process chemicals in a sequential manner.
name: start and stop a specific droplet-chain.
name: target, the chemical which should be delivered in an output-droplet.
name: separate a specific sequence from the rest of the soup.
The compiler thus distinguishes equations (a '→' is part of the command-line) from conditions (a ':' is part of the command-line) and function calls (curved brackets are part of the command-line). The following functions are known to the compiler: extract, insert, halt, loop, mixture, move, select, separate, sequence, start, stop, target, wait. All conditions and equations are subjected to specific droplets and are active in parallel. Only 'sequence' allows specification of sequential processes, the compiler has to assure that the required chemicals are inserted into the specified droplet.

Output-set
Output can be taken from modified input-set droplets (Output A) or from modified register-set droplets (e.g. Output B). It is assumed that these droplets can be captured in a tube and then, after freezing, cut out of the capillary.

Register-set
The register-set is constituted via *n* lanes with empty droplets (these only contain buffer-solutions). Further registers can be added at alternating positions along the meandering lane of input droplets, see firstfigure for illustration.

Processing
Whether the given task can be processed or not depends on the details provided by the rules and the constraints of the given chemicals. The compiler has to solve a satisfiability problem, which can be done exhaustively in simple systems and must be done by heuristics in more complicated endeavours.

Low-level operations
The following list comprises the low-level operations the compiler has at hand to satisfy the given constraints and rules:

- Start and stop of a droplet-chain, all droplet-chains, whether they are the input-set or one of the many possible registers don't matter, they must be stoppable individually.
- Insert material into droplet. The material must be affected by electric fields and it will be extracted from the gel-phase of the transportation and separation network.
- Extract material from a droplet. The material must be affected by electric fields and it will be inserted into the gel-phase of the transportation and separation network.
- Transport material along the transportation and separation network, if this material is sensitive to electric fields.
- Incubate material (wait), simply do nothing at a specific temperature.
- Divert a droplet-chain via a Y-structure and extract some droplets from the chain.
- Merge many droplets in a merging chamber (this is a possible extension of the compiler activities).

- Split droplets with two consecutive droplet generators (this is a possible extension of the compiler activities).
- Separate material in the transportation and separation network. The separation will be done by online detection and assumes that two species will be separated and the slower species being the second one with a clear ditch in between the two separated species.
- Temperature control to allow the implementation of PCR or optimal reaction conditions for the chemical substances.

Ideally the compiler would be able to find a solution to the given specified problem only requiring the user write a set of the three types of chemical rules (canonical form, classifiers and basic operations). The compiler will then produce an intermediate script of low-level commands which still can and perhaps must be edited by the human operator. This script of low-level commands will then be translated by an assembler-stage which produces a RUB-IML script with all the detailed instructions to solve the high-level specified problem, see the BioPRO (RUB-control) manual as a reference for available commands and available operations.

The RUB-IML (InterMediate Language, see WP3 activity report of period 1)

The RUB-IML is an interpreted and run-time compiled language that is intimately coupled to the RUB-control software. It is especially developed to provide a higher-level control of the electrodes activation and the optical detection system. It allows one to incorporate arbitrary many scripts as low-level commands especially using level-one elements like state-machines which act as feedback-controllers. The most important task of these feedback-controllers is to abstract away the physical unknowns and

instance: Create formally an instance as a name of a container. This instance can move along a series of electrodes, e.g. as a droplet, or it is part of another instance.

instance {instance-name}

addprop: Add a property to an instance or chemical. There are certain properties:

addprop control {script-file} {central electrode}
addprop content {list of chemicals or other instances (instances only)}
addprop type ['controller', 'membrane']
addprop permeability {value}
addprop concentration {value}
addprop area {value of polygon area (instances only)}
addprop address {name or identifier of expressed address}

drop: Delete an instance or chemical because it vanished in the experiment or was pushed outside region of interest.

drop {instance-name}

details. The language is based on the following concepts:

The instance construct

In principle, instance can be many things, a local region of solution containing diffusing particles or chemicals, a droplet, a gel bead or a single chemical compound. Its characteristics are specified by the properties added.

The path construct

Our main approach to chemical processing in microfluidic structures is that all chemicals are confined to channels and electrodes are distributed along these channels. To reflect a network of channels the abstraction of a 'path' was invented. A path is a directed list of electrodes which happens to be in a consecutive sequence of fluid channels. The path is an arbitrary complex one-dimensional structure and should not incorporate mesh like structures to allow for a definite start-to-end walk. For CADMAD, it is also conceivable to extend this concept to paths of processing stations along the microfluidic system. It is assumed that also droplets are following a prespecified path. A full set of operators is available for path manipulation: path, append, car, cdr, cons, cur, ele, last, move, next, nr, prev.

Further constructs

Task: All commands inside a task are executed in sequential order. Also blocks- and sub-blocks are treated sequentially. But arbitrary many tasks can be specified. These tasks are processed in a pseudo-parallel fashion. True parallelism cannot be used here because the single available camera has to move to the regions of interest (ROI) and these ROI might reside far apart from each other (in principle, the system could determine whether two or more tasks fit into the same field-of-view of the camera but this is a software-technically difficult task and will be postponed). This means that after a command finished in task A, task B is selected and the next command in that task is executed.

Variables: Variables are instantiated after first naming them in the assign command. Status-codes can be assigned to variables as well as to other variables. Basic arithmetic operations can be realized also as part of the assign-command.

Block: A block starts with the begin-command and finishes at the end-command. A block can contain arbitrary many sub-blocks. All commands in a block are processed in a sequential order. A block is a mere syntactical concept and has no physical counter-part.

Conditions: Three basic conditions are available, GT, GE and EQ. Comparing different types with each other is done implicitly if supported otherwise an error-message is issued.

Control: Further typical commands of a structured programming language are: begin, break, do, elif, else, end, enddo, endif, endtask, exec, if.

Low-Level: Some of the low-level commands to control the underlying hardware are:

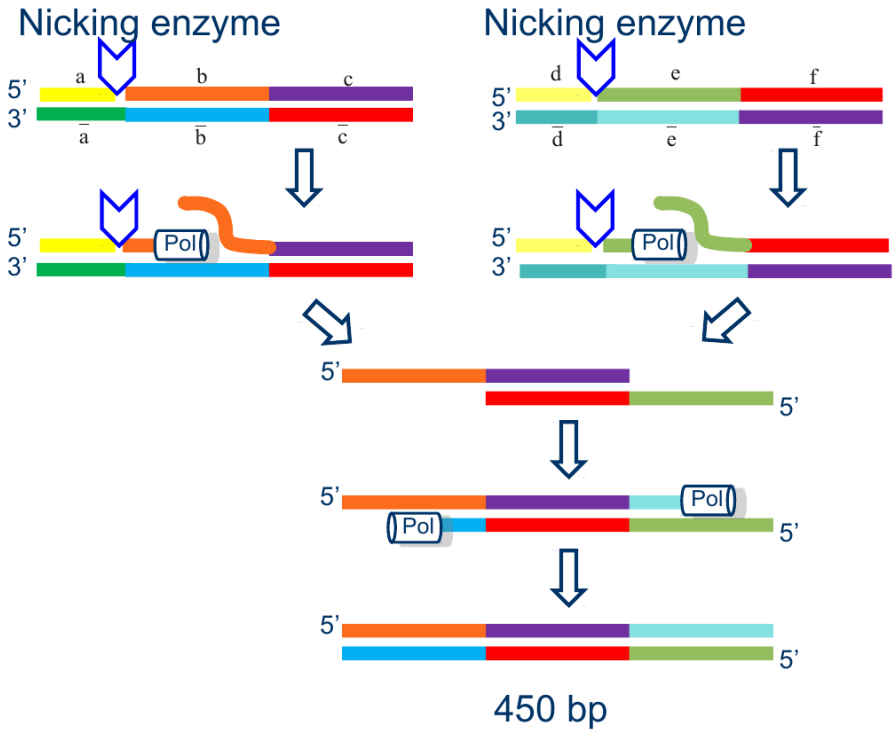
nr_states: Number of states in automaton
add_state: A full state specification to a state-machine
move_back: Move xy-table to last position
move_ori: Move xy-table to original position
move_xy: Move the xy-table to the center of a given electrode.
set_pin: Control the polarity and activity of a pin or electrode.
set_sensor: Specify thresholds and other parameters
zoom: Zoom into a graphical window.

To summarize: high-level constructs like the three types of rules mentioned above are used to formally specify the problem at hand. The task of the compiler (RUB-planner) is to create a set of scripts using RUB-IML. If the compiler has entire knowledge of all the substances this set of RUB-IML scripts can be translated to RUB-control commands which then can be interpreted and executed by the BioPRO-control software realizing the experiments. To illustrate this a little some examples are presented in the following.

Examples

Ex. 1. Y-operation

The Y-operation shown below is a strongly simplified reaction scheme. The rules-set derived from that only shows the principle of doing computation with this compiler and is not able to solve the real problem.



Chemicals

It is assumed that the system is started with two template double-strands dsT1 and dsT2 and the two enzymes. All of them are provided in different reservoirs on the input chip. All the other defined chemicals are intermediate reaction products to allow the chemical rules to be formulated:

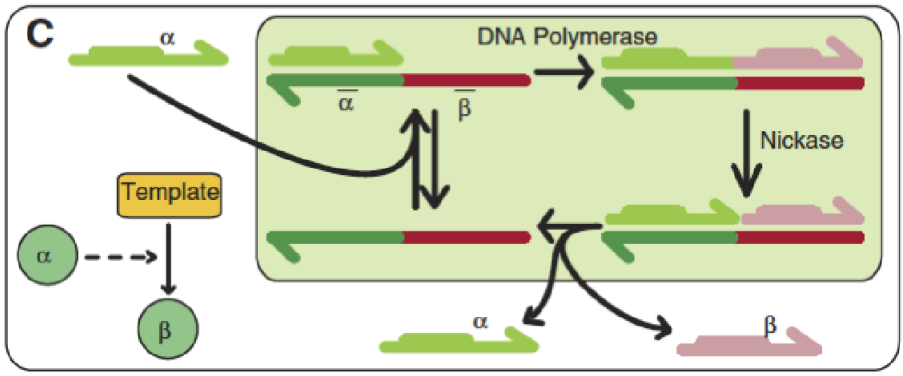
- nicking enzyme NE
- Polymerase POL
- Templates dsT1 = dsABC and dsT2 = dsDEF, sequences ssA, ssBC, ssD, ssEF, ssCBA, ssFED, ssC = revcomp ssF, dsT3 = dsBCE.
-

Rules

The rules only specify the relationships between the particular entities in the system. The optimal processing procedure has to be calculated by the compiler and translated into commands being understood by the underlying control software:

nickA: dsT1 + NE ⟶ ssCBA + ssA + ssBC + NE
nickB: dsT2 + NE ⟶ ssFED + ssD + ssEF + NE
polA: ssCBA + ssA + ssBC + POL ⟶ ssBC + POL
polB: ssFED + ssD + ssEF + POL ⟶ ssEF + POL
polC: ssBC + ssEF + POL ⟶ dsT3 + POL
input1: mixture(dsT1(1mM), NE(1mM))
input2: mixture(dsT2(1mM), NE(1mM))
output: target(dsT3)

Ex. 2 Montagne-System



Montagne, Plasson et.al. Molecular Systems Biology 7:466 2011

Chemicals

System is started with ssT1, ssT2, ssA and two enzymes. All of them are provided in different reservoirs on the input chip. All other defined chemicals are intermediate reaction products to allow the chemical rules to be formulated:

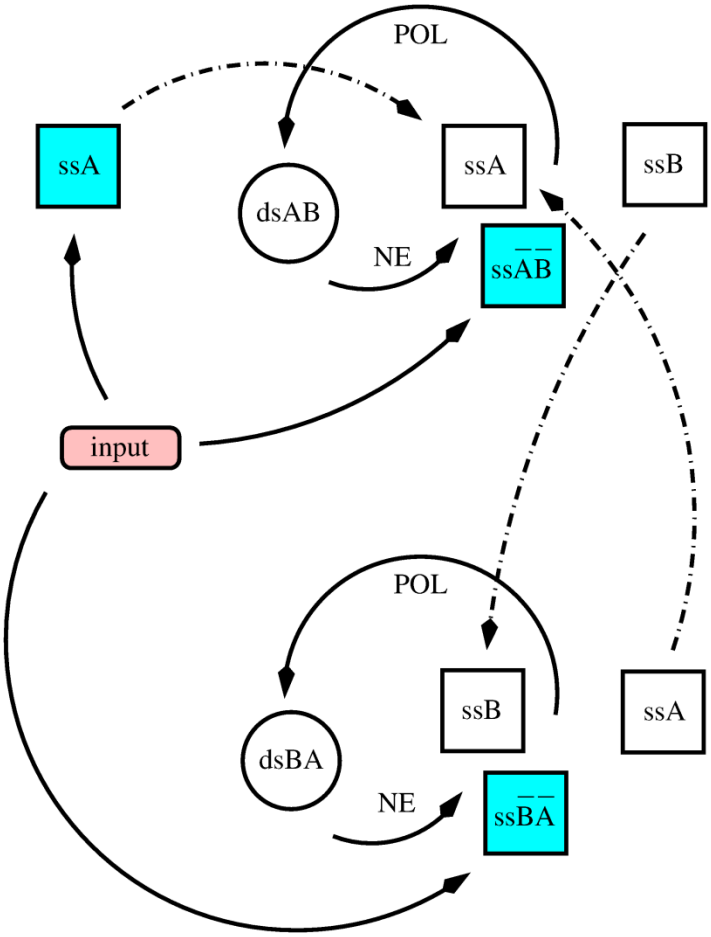
- nicking enzyme NE (Nt.BstNBI)
- Polymerase POL

T1 = 5'-AACAGACTCGA-GCATGACTCAT-3' = ssAB
T1 = 3'-TTGTCTGAGCT-CGTACTGAGTA-5' = ssBA
T2 = 5'-GCATGACTCAT-AACAGACTCGA-3' = ssBA
T2 = 3'-CGTACTGAGTA-TTGTCTGAGCT-5' = ssAB
A = 5'-TCGAGTCTGTT-3' A = 5'-AACAGACTCGA-3'
B = 5'-ATGAGTCATGC-3' B = 5'-GCATGACTCAT-3'

Templates ssT1 = ssAB, ssT2 = ssBA and sequence ssA as a starter. Sequences ssT1 and ssT2 are not consumed. Labeling can be done with an intercalator or specifically labeled templates.

Rules

The reaction graph is shown in the following picture:

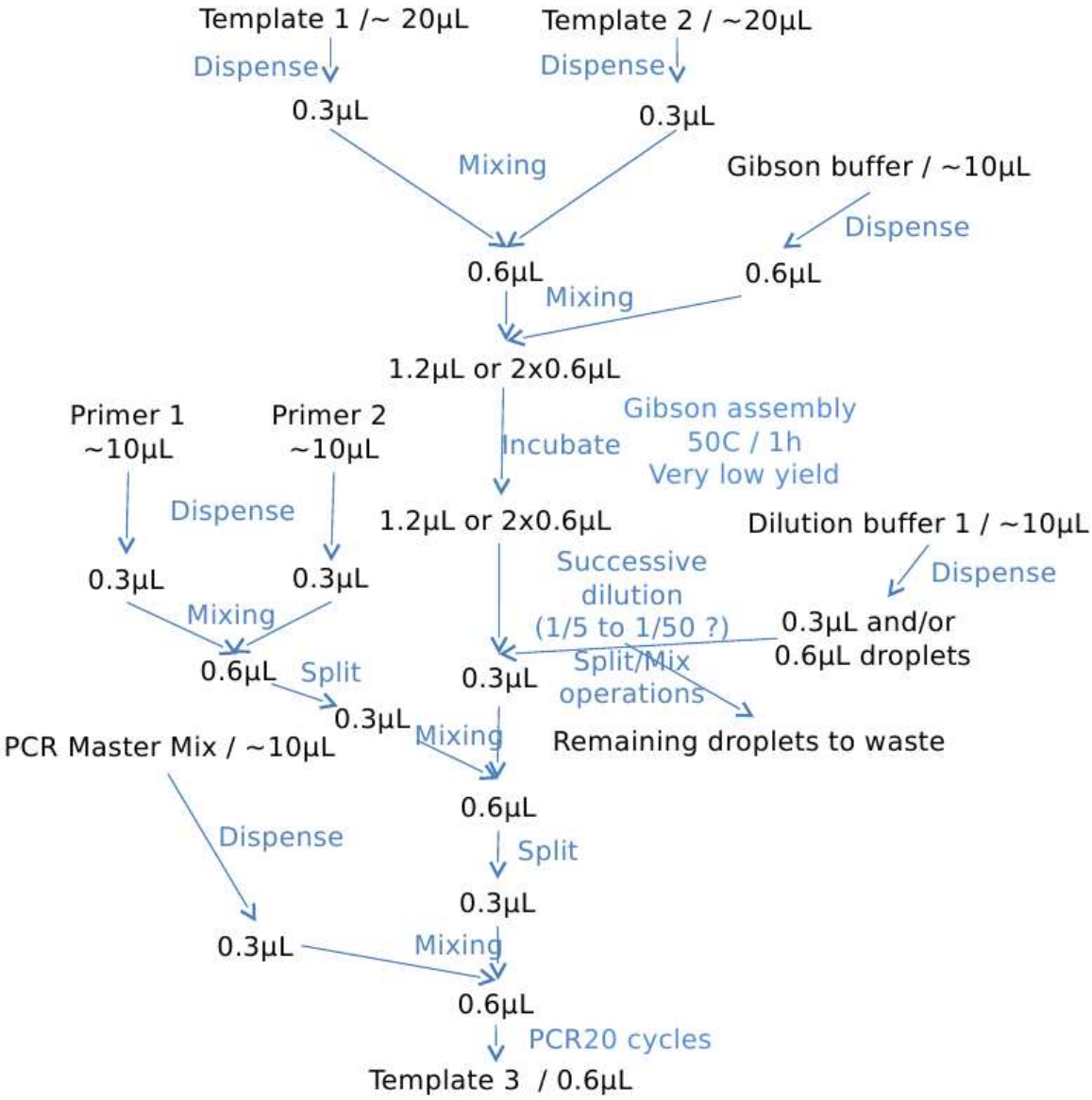


The simplest solution would be to put all ingredients into one droplet. To show the power of the compiler each rule is processed in a distinct droplet:

nickA: dsAB + NE ⟶ ssBA + ssA + ssB + NE
nickB: dsBA + NE ⟶ ssAB + ssB + ssA + NE
polA: ssBA + ssB + POL ⟶ dsAB + POL
polB: ssAB + ssA + POL ⟶ dsBA + POL
separA: separate(ssBA + ssA + ssB, ssA)
separB: separate(ssAB + ssB + ssA, ssB)
sequA: sequence(polA, nickA, separA)
sequB: sequence(polB, nickB, separB)
loopA: loop(10, sequA)
loopB: loop(10, sequB)
output: target(ssA, ssB)

Ex. 3 Y-operation using Gibson

This procedure is provided by Shapiro group at the Weizmann Institute:



Only the first part of the Gibson-procedure is transcribed into a program translated by the compiler.

Chemicals
The detailed sequences are not specified here but typical sequence lengths are in the range of 100 and more nucleotides:

Templates T1, T2, Gibson-Buffer GB, Primers P1, P2, Dilution-Buffer, PCR Master-Mix PCRM.
Further, on the fly created substances:

T12, T12GB, P12, T12GBr, T12GBrd, T12GBs, P12, P12s, TP12, PCR12, PCR, T3, T3O

Beads-Buffer BB, Washing-Buffer WB, Elution-Buffer EB are used in later steps and not documented here.

Rules
T12: mixture(T1(0.3nl), T2(0.3nl))
T12GB: mixture(T12, GB(0.6nl))
T12GBr: wait(3600, 50°C)
T12GBrd: mixture(T12GBr, DB(0.3nl))
T12GBs: select(T12GBrd, 10)
P12: mixture(P1(0.3nl), P2(0.3nl))
P12s: select(P12, 2)
TP12: mixture(T12GBs(0.3nl), P12s(0.3nl))
PCR12: mixture(TP12(0.3nl), PCRM(0.3nl))
PCR: sequence(wait(20, 45°C), wait(20, 55°C), wait(20, 90°C))
T3: loop(20, PCR)
T3O: sequence(T12, T12GB, T12GBr, T12GBrd, T12GBs, P12, P12s, TP12, PCR12,PCR, T3)
output: target(T3O)

In contrast to the original drawing the used amounts have been divided by 1000 because the droplets only can contain around 120 pl. With five droplets of the same substances this would give a volume of 0.6nl. There is a trade-off between the number of processed droplets and the accuracy of used mixtures.

3. Conclusions

The specifications for a compiler for microfluidic integration depends critically on finding a relevant abstraction language for the low level microfluidic functions and on having a good target language for driving the microfluidic system. We are fortunate to have had a good platform for the latter (only requiring extension to droplet operators) in the RUB BioPro Software. The former is comparatively simple for a purely container-based approach involving only miniaturization issues, but this is not sufficient for microfluidic integration. Here we have developed a high-level description language focussing on droplets within microfluidic systems, but recognizing that extraction and injection operations (and the on-chip separation protocol processing in channels between them) must be dealt with. We have shown how such a high level description language applies to several examples in DNA synthesis. The compiler specifications involve mapping this high level language to the RUB BioPro Software Manual which is included as a separate Appendix to this Deliverable.

4. Abbreviations

List all abbreviations used in the document arranged alphabetically.

IML	InterMediate Language
PCR	Polymerase Chain Reaction
ROI	Region Of Interest
RUB-IML	Ruhr Universität Bochum - InterMediate Language

5. References

[i] M. Hucka et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models, BIOINFORMATICS 3139:524–531 2003