ICT Call 7
ROBOHOW.COG
FP7-ICT-288533

Deliverable D3.1:

Documentation about the constraint-based framework

March 21, 2014

Project acronym:	ROBOHOW.COG
Project full title:	Web-enabled and Experience-based Cognitive Robots that Learn Complex Everyday Manipulation Tasks
Work Package:	WP 3
Document number:	D3.1
Document title:	Documentation about the constraint-based framework
Version:	2.0; full rewrite of Year 1 version
Delivery date:	March 21, 2014
Nature:	Report
Dissemination level:	Public (PU)
Authors:	Gianni Borghesan (KU Leuven) Tinne De Laet (KU Leuven) Dominick Vanthienen (KU Leuven) Herman Bruyninckx (KU Leuven)

The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement n°288533 ROBOHOW.COG.

Contents

Summary

This document summarises and structures the experiences of working one year with all partners to establish terminology, concepts, methodology, and work-flow to help them to (i) make use of constraint-based task specification formalism to simplify their robot programming work, and (ii) to consolidate their research results in the form of (formally specifiable) constraints.[1]

The underlying version is a full rewrite of the version delivered after Year 1. The motivations for this new version are two-fold: (i) the earlier version was rated by the reviewers as not sufficiently clear and concise, and (ii) some of the required developments took longer than expected and have only crystalized during Year 2.

An important practical evolution is that the *Stack of Task* has been selected, early in Year 2, as the common *solver* software framework. That means that the "iTaSC solver" will not appear in our communications anymore. CNRS and KUL have already realised common developments on the *Stack of Task* software, such as refactoring of some code, and adding the feature of *implicit constraints* (e.g., the implicit geometric constraint of "keeping frames parallel"). This co-development is continuing also in Year 3.

Chapter 1 introduces constraint-based motion specification, in the generic formulation that (i) unifies the different *prior art* with which the partners entered the RoboHow project, and (ii) presents a systematic architectural approach to connect constraint-based motion specification to knowledge bases and reasoning.

Chapter 2 reports on the work that is done to formalize the motion specification into symbolic representations ("Domain Specific Languages"). This effort of creating DSLs has multiple objectives: making the semantics more clear, making it easier for application developers to specify tasks, and making it easier to connect to reasoners.

[1] As described in Annex 1 of the Grant Agreement.

Chapter 1

The constraint-based approach to robot task specification and execution

1.1 Constraint-based optimization

This Section introduces generic terminology of constraint-based optimization.

Meta model. Here is the mathematical formulation of a constraint-based optimization, in a *domain-independent* way:

task state & domain	$X \in \mathcal{D}$
desired state	X_d
system state & domain	$q \in \mathcal{Q}$
objective function	$\min_q f(X)$
equality constraints	$g(X) = 0$
inequality constraints	$h(X) \leq 0$
tolerances	$d(X, X_d) \leq A$
solver	**algorithm** to compute q

Objective functions, as well as constraints, can be *composed* in two complementary ways:

- *weights*: the composite objective function or constraint is the weighted sum of a set of objective functions or constraints;
- *priorities*: some subset of objective functions or constraints are optimized for *before* another subset. Multiple levels of prioritization can occur.

The *type* of the functions f, g and h determine the *complexity* of the problem, because very different solver algorithms can be required for specific combinations of them.
The function $d(X, X_d)$ is a *distance* function in the state space of the problem, that one needs to specify to determine when the solver of the optimization problem has reached the desired optimal state in a "sufficiently good" way.

Domain model. Applying the generic mathematical formulation in a particular application domain requires the filling in of:

- the *types* for f, X, q,... for that particular *domain*;

- the particular *type* of solvers that fit the specificities of the application.

The following Section will do this for the domain of *robot motion*, and for the *type of tasks* that the RoboHow project needs to consider.

Instances. Each *specific application* then makes the above-mentioned domain model complete by further filling in:

- *parameter values* for f, X,...
- *solver implementation(s)*.

Domain-independent frameworks exist to specify (and solve) constraint-optimization problems, e.g., CasADi (`https://github.com/casadi/casadi/wiki`). There is a continuous tension between using such domain-independent frameworks (because developers then have to learn only one) on the one hand, and, on the other hand, domain-specific versions (that promise the advantage of having, both, more clear semantics for a particular application domain, and more efficient solvers). In RoboHow, we clearly go in the second direction, for both mentioned reasons.

1.2 Constraint-based optimization for robot motion

As mentioned in the previous Section, applying constraint-based optimization to the domain of robot motions requires further specification of:

- *task state & domain*. The RoboHow tasks are "robot motions", this means that "end-effectors" and "links" of robots have to be moved towards specific "target locations", with specific "motion dynamics". The project focus on "knowledge driven" tasks allows to bring strong *structure* in the domain, in that constraints and objective functions come from five different "sources":
  - the *motion capabilities* of the robot devices: each robot has limits in its workspace, actuator power, accuracy, rigidity, etc.
  - the *requirements* from the tasks: baking pancakes can only be done successfully if a recipe is followed, etc.
  - the *affordances* of the objects that the robot manipulates, or navigates in: pieces of fruit must be handled differently than metal utensils; oven plates should not be touched; etc.
  - the properties of the *environment* in which the task is executed: risk involved in hitting objects, or moving through specific areas, etc.
  - *common knowledge*: when grippers are opened, the graspef object might not move together with the robot anymore; kitchens have doors, stoves,...; etc.

  Identifying the sources of constraints, objective functions and tolerances helps in defining the knowledge contexts required to reason about motion tasks.
- *desired state*: these are typically "mechanical": locations, positions, orientations, interaction forces, etc.

- *system state & domain*: depending on the type of robot controller that is available, the constraint optimizer must compute joint positions, joint velocities, joint torques, or motor currents, to be given to the physical actuator system of the robot. In general, each robot system comes with a fixed and limited choice in its *low level* control law, so often the outcome of the constraint optimization is "post-processed", in the sense that it only provides the *desired* motion trajectory, and not the actual actuation directly.

- *objective functions, constraints, tolerances*. The sources of these have already been described above, but here we add some complementary requirements:

  - *sequencing*: even the simplest robot tasks do not consist of one single constrained optimization problem, but requires complex sequences of such sub-tasks. Hence, the constraint-based optimization context that RoboHow is confronted with is that of so-called *hybrid event systems*: also the *transitions* between the optimization problems in different sub-tasks must be dealt with, and preferably in an "optimal" way.

    Two complementary mechanisms are introduced to specificy and execute the required sequencing: (i) Finite State Machines, and (ii) Scheduling.

    A Finite State Machine ("FSM") defines in each of its states what the optimization problem is to be solved in that state, as well as which variables in the system must be *monitored* for reaching certain thresholds that indicate that a transition to another state is required. Defining the FSMs requires a lot of *knowledge*, which fits very nicely in the RoboHow context.

    A Schedule is a data structure, or an algorithm, that defines which optimization solvers to trigger at each moment over time. Typically, they "run" at a much higher rate than the FSMs, for example to select the step gait of a walking humanoid robot, or the execution of an estimator required to provide the values of some parameters in the optimization problem.

  - *implicit vs explicit*. An explicit constraint has the form $y = f(x)$, and has the advantage that its computation is typically straightforward. An implicit constraint has the form $f(x, y) = 0$ and can be more difficult to evaluate since, both, x and y could be "input" or "output" and this can even change during the task.

- *solvers*: RoboHow is coping with all of the above-mentioned aspects, which means that it can cope with a very large variation in tasks, and in intelligent execution of tasks based on sensor information and knowledge sources. In addition, even within the same project, several solver implementations are being developed in parallel. We can identify which use cases are best solved by which solver, and which and how solvers can be combined.

  This flexibility, however, comes at a significant cost: the efforts to design and implement the software for the *solvers* are becoming huge, at the risk of going beyond what one single human can still understand, analyse, document, and synthesize.

Chapter 2

Formalization into Domain-Specific Languages

This Chapter reports on the work that is done to formalize the motion specification into symbolic representations ("Domain Specific Languages"). This effort of creating DSLs has multiple objectives: making the semantics more clear, making it easier for application developers to specify tasks, and making it easier to connect to reasoners.
The main content of this Chapter is provided in an Appendix, which represents the draft of a journal paper that is under construction, exactly for the above-mentioned purposes of describing how and why DSLs have their place in modern, knowledge-driven robot task specification and execution systems. The abstract of this paper is repeated here, for convenience.

The problem of robotic task definition and execution got its first methodological solution in the formalism of Mason, in 1981, that tackles the problem by defining as *setpoint constraints* the position, velocity, and/or forces expressed in one particular *task frame*, and for a 6-DOF robot. Later extensions generalized this approach to

- *multiple* frames,
- *redundant* robots,
- constraints in *other sensor spaces* such as cameras, and
- tracking *trajectory* constraints.

This work describes further extensions to

- *implicit expressions* of constraints between geometric entities (orthogonality, parallelism, distance, angle, ...) in place of *explicit setpoint constraints*,
- a systematic *composition* of constraints,
- *runtime monitoring* of all constraints (that allows for *runtime sequencing* of constraint sets via, for example, a *Finite State Machine* (FSM)), and
- *formal* task descriptions, that can be used by *symbolic reasoners* to plan and analyses tasks.

This means that tasks are seen as ordered groups of constraints to be realised by the robot's motion controller, with, as optional specification, a possible different set of geometric expressions that measures outputs that are not controlled, but that must be monitored because they are relevant to the task evolution. Those monitored expressions raise events that trigger the FSM to make a decision to switch to another task, that is, another ordered group of constraints to execute and monitor.
For all these task specification elements, formal language definitions are introduced, with the aim of providing clear abstractions from the concrete capabilities (hardware as well as control behaviour) of the executing robot platforms. The influence of each specific platform can be added, also in the form of a set of (not task-related) constraints (such as kinematic and actuator limits), to satisfy by the robot's task controller.
When both task and platform constraints are expressed in formal languages with grounded semantics, it will become possible to reason about particular robot-task combinations and evaluate the feasibility of a particular task before trying to execute it.

Chapter 3

Composing tasks and solver software architectures

The DSL efforts mentioned in the previous Chapter, have mainly been performed at KUL, where a complete overhaul of its "iTaSC" formalism is under construction. This effort is not only targeted towards the specification of the languages only, but also towards the better integration with overall application and software architectures. For the latter, the concept of the *System Composition Pattern* has been developed, and several parallel efforts have started to apply the pattern in several use cases, and to adapt the software implementations to support it, in a structured way.

One of the major "lessons learned" from Year 1 was the lack of a sufficiently clear, flexible and structured *methodology* to support the specification and execution of more and more complex constraint-based motion tasks. This problem has been "solved" by the introduction of the *System Composition Pattern*, of Fig. 3.1. This pattern works with any kind of "components" (ROS nodes, Orocos components, or function blocks, but only the latter can really exploit the *hierarchical* composition structure that is the key strength of this Pattern.

The *System Composition Pattern* provides a methodological approach to make complex tasks and systems, by separating explicitly the responsibilities and interactions of: the Composer (to build the system out of connected components), the Coordinator (for the supervision of the system's acitivities), the Computational components (that provide the useful task functionalities, i.c., the constraint-based motion), the Configurators (to put the right "magic numbers" into the Computational components), the Monitor (to check whether the online execution realises all the specified constraints and objective functions, and to fire events for the Coordinator if that is not the case anymore), and the Scheduler (to trigger the right components at the right time).

In the context of RoboHow, the separation of concerns and responsibilities that the System Composition Pattern advocates and supports, brings the following expected added value: each of the concerns and responsibilities requires its own specific set of knowledge and expertise "to get it right", which implies that it can be supported by a specific sub-set of all the RoboHow knowledge base(s) and reasoning infrastructure. In other words, specialising the roles of the software agents allows specialising the context in which knowledge can and must be integrated; such specialisation will, hopefully, reduce the complexity of creating the knowledge bases, implementing the reasoners, implementing the on-line query components, and understanding the workings of the overall system.

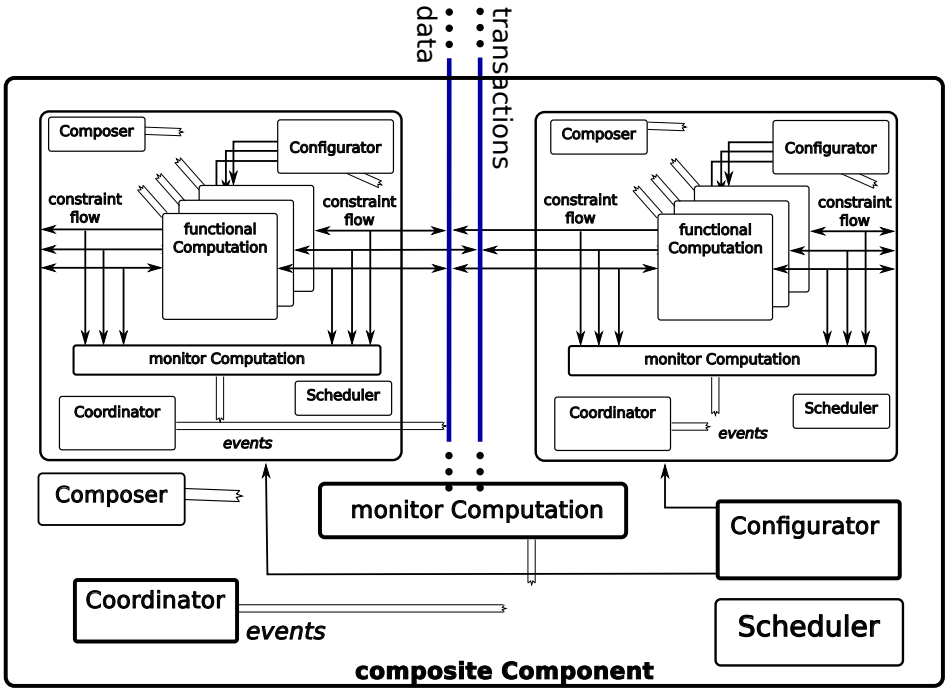

Figure 3.1: The *System Composition Pattern*.

The Pattern has been tested in multiple courses and projects, in the European PhD School in robotics[1], and is underlying several new implementation efforts, such as the *microblx* project.[2] However, the (re)design and (re)implementation efforts required to bring all RoboHow motion specification and control software into the new architectural pattern is huge, and hence still unfinished. Many more efforts will be dedicated to this task in Year 2.

However, it is not just a matter of doing the implementations: the suggested Pattern's advantages have become apparent via the above-mentioned test cases, but switching from a "ROS node"-centric architectural thinking to a "hiearchical composition"-driven thinking requires a lot of retraining. It is also not yet 100% clear which parts of the whole RoboHow software system will really benefit from such a transition.

[1] `http://www.phdschoolinrobotics.eu/`
[2] `https://github.com/kmarkus/microblx`

Chapter 4
Appendix

Draft paper: *Introducing geometric constraint expressions into robot constrained motion specification and control*; Gianni Borghesan, Herman Bruyninckx, KU Leuven, Belgium, 21 March 2014.