Software framework for robot application development: a constraint-based task programming approach

Dominick Vanthienen, Tinne De Laet, Joris De Schutter, Herman Bruyninckx

I. Introduction

When creating a new robot application, how many tasks and platforms are involved that are similar or the same to the ones you used before? Wouldn't it be easy if you could reuse the related parts of the software with minor effort?

In order to address this need, two things are required: first a *systematic approach* for the specification of tasks, and second a *modular software framework*. The instantaneous Task Specification and estimation using Constraints (iTaSC) methodology [2] fulfills the first requirement. This paper presents the iTaSC software framework that fulfills the second requirement.

The iTaSC methodology is a generalization of constraint-based programming that introduces particular sets of auxiliary coordinates to express task constraints and model geometric uncertainty. iTaSC composes multiple tasks, involving *robots* and *objects*, into a *composite task* using weights and priorities. It further describes a composite task as an optimization problem consisting of a set of constraints, possibly expressed in different spaces (different reference frames), and one or more objective functions. *Solving* the optimization problem results in the desired joint values, which are sent to the lower-level robot controllers and hardware, e.g. joint velocities or accelerations. The iTaSC methodology results in a structured workflow to create constraint-based programs for robot applications [3].

The software framework presented in this paper allows a developer to implement an application following the workflow of the iTaSC methodology. In order to obtain modularity and hence reusability, the software framework divides the software needed for an application along two 'orthogonal directions': (i) First, it separates the concerns following the principle of the 5C's [4]–[6] separating the **c**ommunication, **c**omputation, **c**oordination, **c**onfiguration, and **c**omposition functionality; (ii) Second, it regards a robot application at three levels: application, iTaSC, and task level, each of which consist of several components.

The **application level** includes the iTaSC level, as well as other parts that are outside the iTaSC formalism. The **iTaSC level** consists of the elements that form an iTaSC specification, and includes the **task level**, which consists of the functionalities that together form a single task.

All authors are with the Department of Mechanical Engineering, KU Leuven, Belgium. Corresponding author: dominick.vanthienen@mech.kuleuven.be

II. Methods

This section describes the different parts of the iTaSC software framework.

A. Computation

The following paragraphs explain the levels. Each level can be regarded as a composite component, as shown in figure 1. The implementation of the described component layout uses the Orocos component framework [7].

The **application level** components include:

- *Drivers* deliver hardware interfaces.
- *Setpoint generators* deliver desired values for the controllers of iTaSC.

The **iTaSC level** consists of the following components:

- *Robots and objects* contain the kinematic and dynamic models of the robots and objects involved in a robot application. Robots, unlike objects, have controllable degrees-of-freedom (DOF). Their coordinates are denoted $\boldsymbol{q}$.
- The task level *tasks*, as explained in the next paragraph.
- The *solver* is the component that solves the optimization problem consisting of a set of task constraints and one or more objective functions, resulting in the desired values for the low-level robot controllers.
- The *scene* forms the core of the iTaSC composite component, keeping track of the position of the robots and objects in the scene, and between which object frames the tasks are defined.

At **task level**, a task consists of two components:

- The *Virtual Kinematic Chain (VKC)* contains the model of the space between pairs of object frames, defined on robots and objects. It calculates the state of the VKC, the auxiliary (feature) coordinates $\boldsymbol{\chi_f}$, which are required by the constraint-controller.
- The *Constraint-Controller (CC)* implements the output equation $\boldsymbol{y} = f(\boldsymbol{\chi_f}, \boldsymbol{q})$ and the control law that enforces the desired setpoint on the output.

In order to ensure modularity and reusability the components at iTaSC and task level have a fixed structure and port interface, inherited from a base class per type.

B. Coordination

Each level has a state machine that coordinates the behavior at that level. These state machines share the same structure, as shown in figure 2. Each state can be a state machine on its own, as shown for the `Run` state. Together the state machines of all levels form a hierarchical state machine.

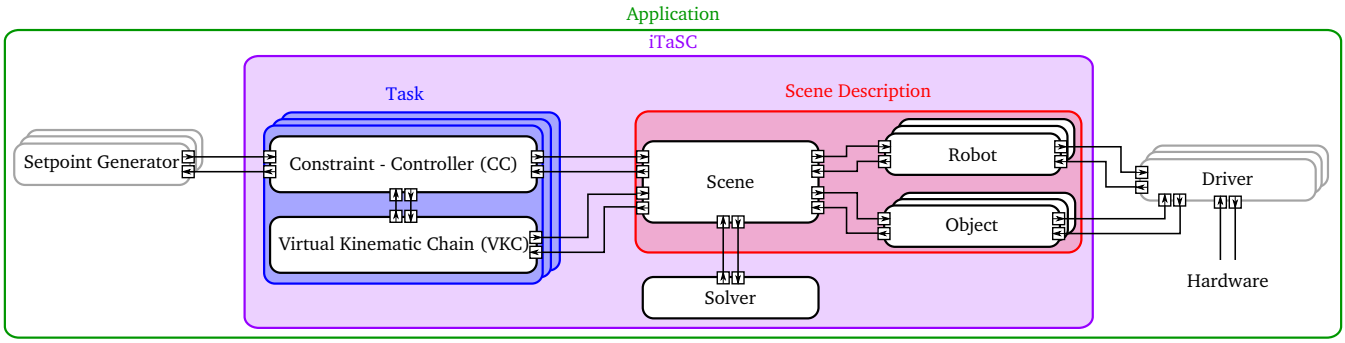

Fig. 1. Computation component layout using sysML flow ports [1]. Stacked boxes refer to possibly multiple components of that type.

These state machines are pure event processors, independent of the software component framework used. They are modeled in rFSM [6], a light-weight, Lua-based statechart model and use the rFSM Orocos framework implementation. Each state machine is loaded in a supervisor component. The supervisor communicates the events of the state machine with the components of that level, and the supervisors of the lower- and higher-level.

C. Configuration, communication and composition

Each of the other concerns needs configuration. The configuration is stored in xml format files and loaded at configuration time.

Ports of the Orocos component framework communicate data and events.

The composition is divided in two parts: a first script `itasc_composition` describes the iTaSC and task level composition, and a second script composes the components at application level.

III. Experiments

An iTaSC application is scalable in terms of number of tasks, robots and objects. An experiment validates this scalability, the modularity, and the reusability. To this end a new application was built consisting of a solver, tasks, robots, and objects already used in earlier published work, such as the human-robot co-manipulation presented at SYROCO in 2012 [3]. The software of the framework and videos of experiments can be found on [8].

IV. Conclusions

This paper presents a modular software framework for constraint-based programming of robot applications, based on the iTaSC methodology. In order to achieve modularity and reusability, it separates concerns following the 5C's approach and separates each of the concerns in independent parts. The paper further presents a reference implementation using the Orocos component framework and rFSM state charts. An experiment validates the scalability, modularity, and reusability.

Future work includes the further decoupling of the iTaSC methodology to a software framework independent library, as well as the development of a Domain Specific Language (DSL) that models an iTaSC based robot application. The DSL will result in an application, soft-and hardware independent programming language for robotics.

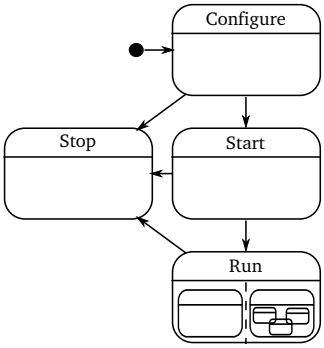

Fig. 2. General structure of a finite state machine.

Acknowledgment

All authors gratefully acknowledge the financial support by the Flemish FWO project G040410N, KU Leuven's Concerted Research Action GOA/2010/011, and European FP7 projects RoboHow (FP7-ICT-288533), BRICS (2008-ICT-231940), and Rosetta (2008-ICT-230902). Tinne De Laet is a PostDoctoral Fellow of the Research Foundation - Flanders (FWO) in Belgium.

References

[1] Object Management Group, "OMG," http://www.omg.org.
[2] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *Int. J. Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007.
[3] D. Vanthienen, T. D. Laet, W. Decré, H. Bruyninckx, and J. D. Schutter, "Force-sensorless and bimanual human-robot comanipulation," in *10th IFAC Symposium on Robot Control (SYROCO)*, vol. 10, Dubrovnik, Croatia, September, 5–7 2012.
[4] M. Radestock and S. Eisenbach, "Coordination in evolving systems," in *Trends in Distributed Systems. CORBA and Beyond*. Springer-Verlag, 1996, pp. 162–176.
[5] E. Prassler, H. Bruyninckx, K. Nilsson, and A. Shakhimardanov, "The use of reuse for designing and manufacturing robots," Robot Standards project, Tech. Rep., 2009, http://www.robot-standards.eu//Documents_RoSta_wiki/whitepaper_reuse.pdf.
[6] M. Klotzbuecher and H. Bruyninckx, "Coordinating robotic tasks and systems with rFSM Statecharts," *J. Software Engin Robotics*, vol. 3, no. 1, pp. 28–56, 2012.
[7] H. Bruyninckx and P. Soetens, "Open RObot COntrol Software (OROCOS)," http://www.orocos.org/, 2001, last visited March 2013.
[8] D. Vanthienen, T. De Laet, R. Smits, and H. Bruyninckx, "itasc software," http://www.orocos.org/itasc, 2011, last visited March 2013.